# DPDK

## DATA PLANE DEVELOPMENT KIT

# DPDK Tools User Guides

### *Release 19.08.0*

**Aug 11, 2019**

# DPDK-PROCINFO APPLICATION

The dpdk-procinfo application is a Data Plane Development Kit (DPDK) application that runs as a DPDK secondary process and is capable of retrieving port statistics, resetting port statistics, printing DPDK memory information and displaying debug information for port. This application extends the original functionality that was supported by dump_cfg.

## 1.1 Running the Application

The application has a number of command line options:

```
./$(RTE_TARGET)/app/dpdk-procinfo -- -m | [-p PORTMASK] [--stats | --xstats |
--stats-reset | --xstats-reset] [ --show-port | --show-tm | --show-crypto |
--show-ring[=name] | --show-mempool[=name] | --iter-mempool=name ]
```

### 1.1.1 Parameters

**-p PORTMASK**: Hexadecimal bitmask of ports to configure.

**--stats** The stats parameter controls the printing of generic port statistics. If no port mask is specified stats are printed for all DPDK ports.

**--xstats** The xstats parameter controls the printing of extended port statistics. If no port mask is specified xstats are printed for all DPDK ports.

**--stats-reset** The stats-reset parameter controls the resetting of generic port statistics. If no port mask is specified, the generic stats are reset for all DPDK ports.

**--xstats-reset** The xstats-reset parameter controls the resetting of extended port statistics. If no port mask is specified xstats are reset for all DPDK ports.

**-m**: Print DPDK memory information.

**--show-port** The show-port parameter displays port level various configuration information associated to RX port queue pair.

**--show-tm** The show-tm parameter displays per port traffic manager settings, current configurations and statistics.

**--show-crypto** The show-crypto parameter displays available cryptodev configurations, settings and stats per node.

**--show-ring[=name]** The show-ring parameter display current allocation of all ring with debug information. Specifying the name allows to display details for specific ring. For invalid or no ring name, whole list is dump.

**–show-mempool[=name]** The show-mempool parameter display current allocation of all mempool debug information. Specifying the name allows to display details for specific specific mempool. For invalid or no mempool name, whole list is dump.

**–iter-mempool=name** The iter-mempool parameter iterates and displays mempool elements specified by name. For invalid or no mempool name no elements are displayed.

## 1.2 Limitations

- dpdk-procinfo should run alongside primary process with same DPDK version.

- When running `dpdk-procinfo` with shared library mode, it is required to pass the same NIC PMD libraries as used for the primary application. Any mismatch in PMD library arguments can lead to undefined behavior and results affecting primary application too.

- Stats retrieval using `dpdk-procinfo` is not supported for virtual devices like PCAP and TAP.

- Since default DPDK EAL arguments for `dpdk-procinfo` are `-c1,-n4 &` `--proc-type=secondary`, It is not expected that the user passes any EAL arguments.

# DPDK-PDUMP APPLICATION

The `dpdk-pdump` tool is a Data Plane Development Kit (DPDK) tool that runs as a DPDK secondary process and is capable of enabling packet capture on dpdk ports.

**Note:**

- The `dpdk-pdump` tool can only be used in conjunction with a primary application which has the packet capture framework initialized already. In dpdk, only the `testpmd` is modified to initialize packet capture framework, other applications remain untouched. So, if the `dpdk-pdump` tool has to be used with any application other than the testpmd, user needs to explicitly modify that application to call packet capture framework initialization code. Refer `app/test-pmd/testpmd.c` code to see how this is done.

- The `dpdk-pdump` tool depends on libpcap based PMD which is disabled by default in the build configuration files, owing to an external dependency on the libpcap development files which must be installed on the board. Once the libpcap development files are installed, the libpcap based PMD can be enabled by setting CONFIG_RTE_LIBRTE_PMD_PCAP=y and recompiling the DPDK.

- The `dpdk-pdump` tool runs as a DPDK secondary process. It exits when the primary application exits.

## 2.1 Running the Application

The tool has a number of command line options:

```
./build/app/dpdk-pdump --
                    [--multi]
                    --pdump '(port=<port id> | device_id=<pci id or vdev name>),
                            (queue=<queue_id>),
                            (rx-dev=<iface or pcap file> |
                             tx-dev=<iface or pcap file>),
                            [ring-size=<ring size>],
                            [mbuf-size=<mbuf data size>],
                            [total-num-mbufs=<number of mbufs>]'
```

The `--multi` command line option is optional argument. If passed, capture will be running on unique cores for all `--pdump` options. If ignored, capture will be running on single core for all `--pdump` options.

The `--pdump` command line option is mandatory and it takes various sub arguments which are described in below section.

---

**Note:**

- Parameters inside the parentheses represents mandatory parameters.

- Parameters inside the square brackets represents optional parameters.

- Multiple instances of `--pdump` can be passed to capture packets on different port and queue combinations.

---

### 2.1.1 The `--pdump` parameters

`port`: Port id of the eth device on which packets should be captured.

`device_id`: PCI address (or) name of the eth device on which packets should be captured.

---

**Note:**

- As of now the `dpdk-pdump` tool cannot capture the packets of virtual devices in the primary process due to a bug in the ethdev library. Due to this bug, in a multi process context, when the primary and secondary have different ports set, then the secondary process (here the `dpdk-pdump` tool) overwrites the `rte_eth_devices[]` entries of the primary process.

---

`queue`: Queue id of the eth device on which packets should be captured. The user can pass a queue value of `*` to enable packet capture on all queues of the eth device.

`rx-dev`: Can be either a pcap file name or any Linux iface.

`tx-dev`: Can be either a pcap file name or any Linux iface.

---

**Note:**

- To receive ingress packets only, `rx-dev` should be passed.

- To receive egress packets only, `tx-dev` should be passed.

- To receive ingress and egress packets separately `rx-dev` and `tx-dev` should both be passed with the different file names or the Linux iface names.

- To receive ingress and egress packets together, `rx-dev` and `tx-dev` should both be passed with the same file name or the same Linux iface name.

---

`ring-size`: Size of the ring. This value is used internally for ring creation. The ring will be used to enqueue the packets from the primary application to the secondary. This is an optional parameter with default size 16384.

`mbuf-size`: Size of the mbuf data. This is used internally for mempool creation. Ideally this value must be same as the primary application's mempool's mbuf data size which is used for packet RX. This is an optional parameter with default size 2176.

---

`total-num-mbufs`: Total number mbufs in mempool. This is used internally for mempool creation. This is an optional parameter with default value 65535.

## 2.2 Example

```
$ sudo ./build/app/dpdk-pdump -l 3 -- --pdump 'port=0,queue=*,rx-dev=/tmp/rx.pcap'
$ sudo ./build/app/dpdk-pdump -l 3,4,5 -- --multi --pdump 'port=0,queue=*,rx-dev=/tmp/rx-1.pcap
```

# DPDK-PMDINFO APPLICATION

The `dpdk-pmdinfo` tool is a Data Plane Development Kit (DPDK) utility that can dump a PMDs hardware support info.

## 3.1 Running the Application

The tool has a number of command line options:

```
dpdk-pmdinfo [-hrtp] [-d <pci id file] <elf-file>

-h, --help              Show a short help message and exit
-r, --raw               Dump as raw json strings
-d FILE, --pcidb=FILE   Specify a pci database to get vendor names from
-t, --table             Output information on hw support as a hex table
-p, --plugindir         Scan dpdk for autoload plugins
```

**Note:**

- Parameters inside the square brackets represents optional parameters.

# DPDK-DEVBIND APPLICATION

The `dpdk-devbind` tool is a Data Plane Development Kit (DPDK) utility that helps binding and unbinding devices from specific drivers. As well as checking their status in that regard.

## 4.1 Running the Application

The tool has a number of command line options:

```
dpdk-devbind [options] DEVICE1 DEVICE2 ....
```

## 4.2 OPTIONS

- `--help,--usage`

    Display usage information and quit

- `-s,--status`

    Print the current status of all known network interfaces. For each device, it displays the PCI domain, bus, slot and function, along with a text description of the device. Depending upon whether the device is being used by a kernel driver, the `igb_uio` driver, or no driver, other relevant information will be displayed: - the Linux interface name e.g. `if=eth0` - the driver being used e.g. `drv=igb_uio` - any suitable drivers not currently using that device e.g. `unused=igb_uio` NOTE: if this flag is passed along with a bind/unbind option, the status display will always occur after the other operations have taken place.

- `-b driver,--bind=driver`

    Select the driver to use or "none" to unbind the device

- `-u,--unbind`

    Unbind a device (Equivalent to `-b none`)

- `--force`

    By default, devices which are used by Linux - as indicated by having routes in the routing table - cannot be modified. Using the `--force` flag overrides this behavior, allowing active links to be forcibly unbound. WARNING: This can lead to loss of network connection and should be used with caution.

> **Warning:** Due to the way VFIO works, there are certain limitations to which devices can be used with VFIO. Mainly it comes down to how IOMMU groups work. Any Virtual Function device can be used with VFIO on its own, but physical devices will require either all ports bound to VFIO, or some of them bound to VFIO while others not being bound to anything at all.
>
> If your device is behind a PCI-to-PCI bridge, the bridge will then be part of the IOMMU group in which your device is in. Therefore, the bridge driver should also be unbound from the bridge PCI device for VFIO to work with devices behind the bridge.

> **Warning:** While any user can run the `dpdk-devbind.py` script to view the status of the network ports, binding or unbinding network ports requires root privileges.

## 4.3 Examples

To display current device status:

```
dpdk-devbind --status
```

To bind eth1 from the current driver and move to use igb_uio:

```
dpdk-devbind --bind=igb_uio eth1
```

To unbind 0000:01:00.0 from using any driver:

```
dpdk-devbind -u 0000:01:00.0
```

To bind 0000:02:00.0 and 0000:02:00.1 to the ixgbe kernel driver:

```
dpdk-devbind -b ixgbe 02:00.0 02:00.1
```

To check status of all network ports, assign one to the igb_uio driver and check status again:

```
# Check the status of the available devices.
dpdk-devbind --status
Network devices using DPDK-compatible driver
============================================
<none>

Network devices using kernel driver
===================================
0000:0a:00.0 '82599ES 10-Gigabit' if=eth2 drv=ixgbe unused=


# Bind the device to igb_uio.
sudo dpdk-devbind -b igb_uio 0000:0a:00.0


# Recheck the status of the devices.
dpdk-devbind --status
Network devices using DPDK-compatible driver
============================================
0000:0a:00.0 '82599ES 10-Gigabit' drv=igb_uio unused=
```

# DPDK-TEST-BBDEV APPLICATION

The `dpdk-test-bbdev` tool is a Data Plane Development Kit (DPDK) utility that allows measuring performance parameters of PMDs available in the bbdev framework. Available tests available for execution are: latency, throughput, validation and sanity tests. Execution of tests can be customized using various parameters passed to a python running script.

## 5.1 Compiling the Application

**Step 1: PMD setting**

The `dpdk-test-bbdev` tool depends on crypto device drivers PMD which are disabled by default in the build configuration file `common_base`. The bbdevice drivers PMD which should be tested can be enabled by setting

```
CONFIG_RTE_LIBRTE_PMD_<name>=y
```

Setting example for (*baseband_turbo_sw*) PMD

```
CONFIG_RTE_LIBRTE_PMD_BBDEV_TURBO_SW=y
```

**Step 2: Build the application**

Execute the `dpdk-setup.sh` script to build the DPDK library together with the `dpdk-test-bbdev` application.

Initially, the user must select a DPDK target to choose the correct target type and compiler options to use when building the libraries. The user must have all libraries, modules, updates and compilers installed in the system prior to this, as described in the earlier chapters in this Getting Started Guide.

## 5.2 Running the Application

The tool application has a number of command line options:

```
python test-bbdev.py [-h] [-p TESTAPP_PATH] [-e EAL_PARAMS] [-t TIMEOUT]
                     [-c TEST_CASE [TEST_CASE ...]]
                     [-v TEST_VECTOR [TEST_VECTOR...]] [-n NUM_OPS]
                     [-b BURST_SIZE [BURST_SIZE ...]] [-l NUM_LCORES]
```

### 5.2.1 command-line Options

The following are the command-line options:

**-h,--help** Shows help message and exit.

**-p TESTAPP_PATH,--testapp_path TESTAPP_PATH** Indicates the path to the bbdev test
app. If not specified path is set based on *$RTE_SDK* environment variable concatenated with
"*/build/app/testbbdev*".

**-e EAL_PARAMS,--eal_params EAL_PARAMS** Specifies EAL arguments which are passed to
the test app. For more details, refer to DPDK documentation at http://doc.dpdk.org/guides/linux_
gsg/linux_eal_parameters.html.

**-t TIMEOUT,--timeout TIMEOUT** Specifies timeout in seconds. If not specified timeout is set
to 300 seconds.

**-c TEST_CASE [TEST_CASE ...],--test_cases TEST_CASE [TEST_CASE ...]**
Defines test cases to run. If not specified all available tests are run.

> **Example usage:**
>
> **./test-bbdev.py -c validation** Runs validation test suite
>
> **./test-bbdev.py -c latency throughput** Runs latency and throughput test suites

**-v TEST_VECTOR [TEST_VECTOR ...],--test_vector TEST_VECTOR [TEST_VECTOR ...]**
Specifies paths to the test vector files. If not specified path is set based on *$RTE_SDK* environ-
ment variable concatenated with "*/app/test-bbdev/test_vectors/bbdev_null.data*" and indicates
default data file.

> **Example usage:**
>
> **./test-bbdev.py -v app/test-bbdev/test_vectors/turbo_dec_test1.data**
> Fills vector based on turbo_dec_test1.data file and runs all tests
>
> **./test-bbdev.py -v turbo_dec_test1.data turbo_enc_test2.data** The
> bbdev test app is executed twice. First time vector is filled based on *turbo_dec_test1.data*
> file and second time based on *turb_enc_test2.data* file. For both executions all tests are run.

**-n NUM_OPS,--num_ops NUM_OPS** Specifies number of operations to process on device. If not
specified num_ops is set to 32 operations.

**-l NUM_LCORES,--num_lcores NUM_LCORES** Specifies number of lcores to run. If not speci-
fied num_lcores is set according to value from RTE configuration (EAL coremask)

**-b BURST_SIZE [BURST_SIZE ...],--burst-size BURST_SIZE [BURST_SIZE ...]**
Specifies operations enqueue/dequeue burst size. If not specified burst_size is set to 32. Maximum
is 512.

## 5.2.2 Test Cases

There are 6 main test cases that can be executed using testbbdev tool:

- **Sanity checks [-c unittest]**

  – Performs sanity checks on BBDEV interface, validating basic functionality

- **Validation tests [-c validation]**

  – Performs full operation of enqueue and dequeue

  – Compares the dequeued data buffer with a expected values in the test vector (TV) being
  used

– Fails if any dequeued value does not match the data in the TV

- **Offload Cost measurement [-c offload]**

  – Measures the CPU cycles consumed from the receipt of a user enqueue until it is put on the device queue

  – **The test measures 4 metrics**

    1. *SW Enq Offload Cost*: Software only enqueue offload cost, the cycle counts and time (us) from the point the enqueue API is called until the point the operation is put on the accelerator queue.

    2. *Acc Enq Offload Cost*: The cycle count and time (us) from the point the operation is put on the accelerator queue until the return from enqueue.

    3. *SW Deq Offload Cost*: Software dequeue cost, the cycle counts and time (us) consumed to dequeue one operation.

    4. *Empty Queue Enq Offload Cost*: The cycle count and time (us) consumed to dequeue from an empty queue.

- **Latency measurement [-c latency]**

  – Measures the time consumed from the first enqueue until the first appearance of a dequeued result

  – This measurement represents the full latency of a bbdev operation (encode or decode) to execute

- **Poll-mode Throughput measurement [-c throughput]**

  – Performs full operation of enqueue and dequeue

  – Executes in poll mode

  – Measures the achieved throughput on a subset or all available CPU cores

  – Dequeued data is not validated against expected values stored in TV

  – Results are printed in million operations per second and million bits per second

- **Interrupt-mode Throughput [-c interrupt]**

  – Similar to Throughput test case, but using interrupts. No polling.

### 5.2.3 Parameter Globbing

Thanks to the globbing functionality in python test-bbdev.py script allows to run tests with different set of vector files without giving all of them explicitly.

**Example usage:**

```
./test-bbdev.py -v app/test-bbdev/test_vectors/turbo_<enc/dec>_c<c>_k<k>_r<r>_e<e>_<extra-info>
```

It runs all tests with following vectors:

- `bbdev_null.data`

- `turbo_dec_c1_k6144_r0_e34560_sbd_negllr.data`

- `turbo_enc_c1_k40_r0_e1196_rm.data`

- `turbo_enc_c2_k5952_r0_e17868_crc24b.data`

- `turbo_dec_c1_k40_r0_e17280_sbd_negllr.data`

- `turbo_dec_c1_k6144_r0_e34560_sbd_posllr.data`

- `turbo_enc_c1_k40_r0_e272_rm.data`

- `turbo_enc_c3_k4800_r2_e14412_crc24b.data`

- `turbo_dec_c1_k6144_r0_e10376_crc24b_sbd_negllr_high_snr.data`

- `turbo_dec_c2_k3136_r0_e4920_sbd_negllr_crc24b.data`

- `turbo_enc_c1_k6144_r0_e120_rm_rvidx.data`

- `turbo_enc_c4_k4800_r2_e14412_crc24b.data`

- `turbo_dec_c1_k6144_r0_e10376_crc24b_sbd_negllr_low_snr.data`

- `turbo_dec_c2_k3136_r0_e4920_sbd_negllr.data`

- `turbo_enc_c1_k6144_r0_e18444.data`

- `turbo_dec_c1_k6144_r0_e34560_negllr.data`

- `turbo_enc_c1_k40_r0_e1190_rm.data`

- `turbo_enc_c1_k6144_r0_e18448_crc24a.data`

- `turbo_dec_c1_k6144_r0_e34560_posllr.data`

- `turbo_enc_c1_k40_r0_e1194_rm.data`

- `turbo_enc_c1_k6144_r0_e32256_crc24b_rm.data`

`./test-bbdev.py -v app/test-bbdev/turbo_*_default.data`

It runs all tests with "default" vectors.

- `turbo_dec_default.data` is a soft link to `turbo_dec_c1_k6144_r0_e10376_crc24b_sbd_negl`

- `turbo_enc_default.data` is a soft link to `turbo_enc_c1_k6144_r0_e32256_crc24b_rm.data`

## 5.3 Running Tests

Shortened tree of isg_cid-wireless_dpdk_ae with dpdk compiled for x86_64-native-linux-icc target:

```
|-- app
    |-- test-bbdev
        |-- test_vectors
            |-- bbdev_null.data
            |-- turbo_dec_c1_k6144_r0_e34560_sbd_negllr.data
            |-- turbo_enc_c1_k40_r0_e1196_rm.data
            |-- turbo_enc_c2_k5952_r0_e17868_crc24b.data
            |-- turbo_dec_c1_k40_r0_e17280_sbd_negllr.data
            |-- turbo_dec_c1_k6144_r0_e34560_sbd_posllr.data
            |-- turbo_enc_c1_k40_r0_e272_rm.data
            |-- turbo_enc_c3_k4800_r2_e14412_crc24b.data
            |-- turbo_dec_c1_k6144_r0_e10376_crc24b_sbd_negllr_high_snr.data
            |-- turbo_dec_c2_k3136_r0_e4920_sbd_negllr_crc24b.data
            |-- turbo_enc_c1_k6144_r0_e120_rm_rvidx.data
            |-- turbo_enc_c4_k4800_r2_e14412_crc24b.data
            |-- turbo_dec_c1_k6144_r0_e10376_crc24b_sbd_negllr_low_snr.data
```

```
            |-- turbo_dec_c2_k3136_r0_e4920_sbd_negllr.data
            |-- turbo_enc_c1_k6144_r0_e18444.data
            |-- turbo_dec_c1_k6144_r0_e34560_negllr.data
            |-- turbo_enc_c1_k40_r0_e1190_rm.data
            |-- turbo_enc_c1_k6144_r0_e18448_crc24a.data
            |-- turbo_dec_c1_k6144_r0_e34560_posllr.data
            |-- turbo_enc_c1_k40_r0_e1194_rm.data
            |-- turbo_enc_c1_k6144_r0_e32256_crc24b_rm.data

|-- x86_64-native-linux-icc
    |-- app
        |-- testbbdev
```

### 5.3.1 All bbdev devices

```
./test-bbdev.py -p ../../x86_64-native-linux-icc/app/testbbdev
-v turbo_dec_default.data
```

It runs all available tests using the test vector filled based on *turbo_dec_default.data* file. By default number of operations to process on device is set to 32, timeout is set to 300s and operations enqueue/dequeue burst size is set to 32. Moreover a bbdev (*baseband_null*) device will be created.

### 5.3.2 baseband turbo_sw device

```
./test-bbdev.py -p ../../x86_64-native-linux-icc/app/testbbdev
-e="--vdev=baseband_turbo_sw" -t 120 -c validation
-v ./test_vectors/turbo_* -n 64 -b 8 32
```

It runs **validation** test for each vector file that matches the given pattern. Number of operations to process on device is set to 64 and operations timeout is set to 120s and enqueue/dequeue burst size is set to 8 and to 32. Moreover a bbdev (*baseband_turbo_sw*) device will be created.

### 5.3.3 bbdev null device

Executing bbdev null device with *bbdev_null.data* helps in measuring the overhead introduced by the bbdev framework.

```
./test-bbdev.py -e="--vdev=baseband_null0"
-v ./test_vectors/bbdev_null.data
```

**Note:**

baseband_null device does not have to be defined explicitly as it is created by default.

## 5.4 Test Vector files

Test Vector files contain the data which is used to set turbo decoder/encoder parameters and buffers for validation purpose. New test vector files should be stored in `app/test-bbdev/test_vectors/` directory. Detailed description of the syntax of the test vector files is in the following section.

### 5.4.1 Basic principles for test vector files

Line started with # is treated as a comment and is ignored.

---

If variable is a chain of values, values should be separated by a comma. If assignment is split into several lines, each line (except the last one) has to be ended with a comma. There is no comma after last value in last line. Correct assignment should look like the following:

```
variable =
value, value, value, value,
value, value
```

In case where variable is a single value correct assignment looks like the following:

```
variable =
value
```

Length of chain variable is calculated by parser. Can not be defined explicitly.

Variable op_type has to be defined as a first variable in file. It specifies what type of operations will be executed. For decoder op_type has to be set to RTE_BBDEV_OP_TURBO_DEC and for encoder to RTE_BBDEV_OP_TURBO_ENC.

Full details of the meaning and valid values for the below fields are documented in *rte_bbdev_op.h*

### 5.4.2 Turbo decoder test vectors template

For turbo decoder it has to be always set to RTE_BBDEV_OP_TURBO_DEC

```
op_type =
RTE_BBDEV_OP_TURBO_DEC
```

Chain of uint32_t values. Note that it is possible to define more than one input/output entries which will result in chaining two or more data structures for *segmented Transport Blocks*

```
input0 =
0x00000000, 0x7f817f00, 0x7f7f8100, 0x817f8100, 0x81008100, 0x7f818100, 0x81817f00, 0x7f818100,
0x81007f00, 0x7f818100, 0x817f8100, 0x81817f00, 0x81008100, 0x817f7f00, 0x7f7f8100, 0x81817f00
```

Chain of uint32_t values

```
input1 =
0x7f7f0000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
```

Chain of uint32_t values

```
input2 =
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
```

Chain of uint32_t values

```
hard_output0 =
0xa7d6732e
```

Chain of uint32_t values

```
hard_output1 =
0xa61
```

Chain of uint32_t values

```
soft_output0 =
0x817f817f, 0x7f817f7f, 0x81818181, 0x817f7f81, 0x7f818181, 0x8181817f, 0x817f817f, 0x8181817f
```

Chain of uint32_t values

```
soft_output1 =
0x817f7f81, 0x7f7f7f81, 0x7f7f8181
```

uint32_t value

```
e =
44
```

uint16_t value

```
k =
40
```

uint8_t value

```
rv_index =
0
```

uint8_t value

```
iter_max =
8
```

uint8_t value

```
iter_min =
4
```

uint8_t value

```
expected_iter_count =
8
```

uint8_t value

```
ext_scale =
15
```

uint8_t value

```
num_maps =
0
```

Chain of flags for turbo decoder operation. Following flags can be used:

- `RTE_BBDEV_TURBO_SUBBLOCK_DEINTERLEAVE`

- `RTE_BBDEV_TURBO_CRC_TYPE_24B`

- `RTE_BBDEV_TURBO_EQUALIZER`

- `RTE_BBDEV_TURBO_SOFT_OUT_SATURATE`

- `RTE_BBDEV_TURBO_HALF_ITERATION_EVEN`

- `RTE_BBDEV_TURBO_CONTINUE_CRC_MATCH`

- `RTE_BBDEV_TURBO_SOFT_OUTPUT`

- `RTE_BBDEV_TURBO_EARLY_TERMINATION`

- `RTE_BBDEV_TURBO_DEC_INTERRUPTS`

- `RTE_BBDEV_TURBO_POS_LLR_1_BIT_IN`

- `RTE_BBDEV_TURBO_NEG_LLR_1_BIT_IN`

- `RTE_BBDEV_TURBO_POS_LLR_1_BIT_SOFT_OUT`

- `RTE_BBDEV_TURBO_NEG_LLR_1_BIT_SOFT_OUT`

- `RTE_BBDEV_TURBO_MAP_DEC`

---

**5.4. Test Vector files**

Example:

```
op_flags =
RTE_BBDEV_TURBO_SUBBLOCK_DEINTERLEAVE, RTE_BBDEV_TURBO_EQUALIZER,
RTE_BBDEV_TURBO_SOFT_OUTPUT
```

Chain of operation statuses that are expected after operation is performed. Following statuses can be used:

- `DMA`

- `FCW`

- `CRC`

- `OK`

`OK` means no errors are expected. Cannot be used with other values.

```
expected_status =
FCW, CRC
```

### 5.4.3 Turbo encoder test vectors template

For turbo encoder it has to be always set to `RTE_BBDEV_OP_TURBO_ENC`

```
op_type =
RTE_BBDEV_OP_TURBO_ENC
```

Chain of uint32_t values

```
input0 =
0x11d2bcac, 0x4d
```

Chain of uint32_t values

```
output0 =
0xd2399179, 0x640eb999, 0x2cbaf577, 0xaf224ae2, 0x9d139927, 0xe6909b29,
0xa25b7f47, 0x2aa224ce, 0x79f2
```

uint32_t value

```
e =
272
```

uint16_t value

```
k =
40
```

uint16_t value

```
ncb =
192
```

uint8_t value

```
rv_index =
0
```

Chain of flags for turbo encoder operation. Following flags can be used:

- `RTE_BBDEV_TURBO_RV_INDEX_BYPASS`

- `RTE_BBDEV_TURBO_RATE_MATCH`

- `RTE_BBDEV_TURBO_CRC_24B_ATTACH`

---

- RTE_BBDEV_TURBO_CRC_24A_ATTACH

- RTE_BBDEV_TURBO_ENC_SCATTER_GATHER

RTE_BBDEV_TURBO_ENC_SCATTER_GATHER is used to indicate the parser to force the input data to be memory split and formed as a segmented mbuf.

```
op_flags =
RTE_BBDEV_TURBO_RATE_MATCH
```

Chain of operation statuses that are expected after operation is performed. Following statuses can be used:

- DMA

- FCW

- OK

OK means no errors are expected. Cannot be used with other values.

```
expected_status =
OK
```

# DPDK-TEST-CRYPTO-PERF APPLICATION

The `dpdk-test-crypto-perf` tool is a Data Plane Development Kit (DPDK) utility that allows measuring performance parameters of PMDs available in the crypto tree. There are available two measurement types: throughput and latency. User can use multiply cores to run tests on but only one type of crypto PMD can be measured during single application execution. Cipher parameters, type of device, type of operation and chain mode have to be specified in the command line as application parameters. These parameters are checked using device capabilities structure.

## 6.1 Limitations

On hardware devices the cycle-count doesn't always represent the actual offload cost. The cycle-count only represents the offload cost when the hardware accelerator is not fully loaded, when loaded the cpu cycles freed up by the offload are still consumed by the test tool and included in the cycle-count. These cycles are consumed by retries and inefficient API calls enqueuing and dequeuing smaller bursts than specified by the cmdline parameter. This results in a larger cycle-count measurement and should not be interpreted as an offload cost measurement. Using "pmd-cyclecount" mode will give a better idea of actual costs of hardware acceleration.

On hardware devices the throughput measurement is not necessarily the maximum possible for the device, e.g. it may be necessary to use multiple cores to keep the hardware accelerator fully loaded and so measure maximum throughput.

## 6.2 Compiling the Application

**Step 1: PMD setting**

The `dpdk-test-crypto-perf` tool depends on crypto device drivers PMD which are disabled by default in the build configuration file `common_base`. The crypto device drivers PMD which should be tested can be enabled by setting:

```
CONFIG_RTE_LIBRTE_PMD_<name>=y
```

Setting example for open ssl PMD:

```
CONFIG_RTE_LIBRTE_PMD_OPENSSL=y
```

**Step 2: Linearization setting**

It is possible linearized input segmented packets just before crypto operation for devices which doesn't support scatter-gather, and allows to measure performance also for this use case.

To set on the linearization options add below definition to the `cperf_ops.h` file:

```
#define CPERF_LINEARIZATION_ENABLE
```

**Step 3: Build the application**

Execute the `dpdk-setup.sh` script to build the DPDK library together with the `dpdk-test-crypto-perf` application.

Initially, the user must select a DPDK target to choose the correct target type and compiler options to use when building the libraries. The user must have all libraries, modules, updates and compilers installed in the system prior to this, as described in the earlier chapters in this Getting Started Guide.

## 6.3 Running the Application

The tool application has a number of command line options:

```
dpdk-test-crypto-perf [EAL Options] -- [Application Options]
```

### 6.3.1 EAL Options

The following are the EAL command-line options that can be used in conjunction with the `dpdk-test-crypto-perf` application. See the DPDK Getting Started Guides for more information on these options.

- `-c <COREMASK>` or `-l <CORELIST>`

    Set the hexadecimal bitmask of the cores to run on. The corelist is a list cores to use.

- `-w <PCI>`

    Add a PCI device in white list.

- `--vdev <driver><id>`

    Add a virtual device.

### 6.3.2 Application Options

The following are the application command-line options:

- `--ptest type`

    Set test type, where `type` is one of the following:

    ```
    throughput
    latency
    verify
    pmd-cyclecount
    ```

- `--silent`

    Disable options dump.

- `--pool-sz <n>`

    Set the number of mbufs to be allocated in the mbuf pool.

- `--total-ops <n>`

    Set the number of total operations performed.

- `--burst-sz <n>`

    Set the number of packets per burst.

    **This can be set as:**

    - Single value (i.e. `--burst-sz 16`)

    - Range of values, using the following structure `min:inc:max`, where `min` is minimum size, `inc` is the increment size and `max` is the maximum size (i.e. `--burst-sz 16:2:32`)

    - List of values, up to 32 values, separated in commas (i.e. `--burst-sz 16,24,32`)

- `--buffer-sz <n>`

    Set the size of single packet (plaintext or ciphertext in it).

    **This can be set as:**

    - Single value (i.e. `--buffer-sz 16`)

    - Range of values, using the following structure `min:inc:max`, where `min` is minimum size, `inc` is the increment size and `max` is the maximum size (i.e. `--buffer-sz 16:2:32`)

    - List of values, up to 32 values, separated in commas (i.e. `--buffer-sz 32,64,128`)

- `--imix <n>`

    Set the distribution of packet sizes.

    A list of weights must be passed, containing the same number of items than buffer-sz, so each item in this list will be the weight of the packet size on the same position in the buffer-sz parameter (a list have to be passed in that parameter).

    Example:

    To test a distribution of 20% packets of 64 bytes, 40% packets of 100 bytes and 40% packets of 256 bytes, the command line would be: `--buffer-sz 64,100,256 --imix 20,40,40`. Note that the weights do not have to be percentages, so using `--imix 1,2,2` would result in the same distribution

- `--segment-sz <n>`

    Set the size of the segment to use, for Scatter Gather List testing. By default, it is set to the size of the maximum buffer size, including the digest size, so a single segment is created.

- `--devtype <name>`

    Set device type, where `name` is one of the following:

    ```
    crypto_null
    crypto_aesni_mb
    crypto_aesni_gcm
    crypto_openssl
    crypto_qat
    crypto_snow3g
    crypto_kasumi
    crypto_zuc
    ```

**6.3. Running the Application**  20

```
crypto_dpaa_sec
crypto_dpaa2_sec
crypto_armv8
crypto_scheduler
crypto_mvsam
```

- `--optype <name>`

    Set operation type, where `name` is one of the following:

    ```
    cipher-only
    auth-only
    cipher-then-auth
    auth-then-cipher
    aead
    ```

    For GCM/CCM algorithms you should use aead flag.

- `--sessionless`

    Enable session-less crypto operations mode.

- `--out-of-place`

    Enable out-of-place crypto operations mode.

- `--test-file <name>`

    Set test vector file path. See the Test Vector File chapter.

- `--test-name <name>`

    Set specific test name section in the test vector file.

- `--cipher-algo <name>`

    Set cipher algorithm name, where `name` is one of the following:

    ```
    3des-cbc
    3des-ecb
    3des-ctr
    aes-cbc
    aes-ctr
    aes-ecb
    aes-f8
    aes-xts
    arc4
    null
    kasumi-f8
    snow3g-uea2
    zuc-eea3
    ```

- `--cipher-op <mode>`

    Set cipher operation mode, where `mode` is one of the following:

    ```
    encrypt
    decrypt
    ```

- `--cipher-key-sz <n>`

    Set the size of cipher key.

- `--cipher-iv-sz <n>`

    Set the size of cipher iv.

- `--auth-algo <name>`

  Set authentication algorithm name, where `name` is one of the following:

  ```
  3des-cbc
  aes-cbc-mac
  aes-cmac
  aes-gmac
  aes-xcbc-mac
  md5
  md5-hmac
  sha1
  sha1-hmac
  sha2-224
  sha2-224-hmac
  sha2-256
  sha2-256-hmac
  sha2-384
  sha2-384-hmac
  sha2-512
  sha2-512-hmac
  kasumi-f9
  snow3g-uia2
  zuc-eia3
  ```

- `--auth-op <mode>`

  Set authentication operation mode, where `mode` is one of the following:

  ```
  verify
  generate
  ```

- `--auth-key-sz <n>`

  Set the size of authentication key.

- `--auth-iv-sz <n>`

  Set the size of auth iv.

- `--aead-algo <name>`

  Set AEAD algorithm name, where `name` is one of the following:

  ```
  aes-ccm
  aes-gcm
  ```

- `--aead-op <mode>`

  Set AEAD operation mode, where `mode` is one of the following:

  ```
  encrypt
  decrypt
  ```

- `--aead-key-sz <n>`

  Set the size of AEAD key.

- `--aead-iv-sz <n>`

  Set the size of AEAD iv.

- `--aead-aad-sz <n>`

  Set the size of AEAD aad.

- `--digest-sz <n>`

**6.3. Running the Application**                                                                    **22**

Set the size of digest.

- `--desc-nb <n>`

    Set number of descriptors for each crypto device.

- `--pmd-cyclecount-delay-ms <n>`

    Add a delay (in milliseconds) between enqueue and dequeue in pmd-cyclecount benchmarking mode (useful when benchmarking hardware acceleration).

- `--csv-friendly`

    Enable test result output CSV friendly rather than human friendly.

### 6.3.3 Test Vector File

The test vector file is a text file contain information about test vectors. The file is made of the sections. The first section doesn't have header. It contain global information used in each test variant vectors - typically information about plaintext, ciphertext, cipher key, auth key, initial vector. All other sections begin header. The sections contain particular information typically digest.

**Format of the file:**

Each line beginning with sign '#' contain comment and it is ignored by parser:

```
# <comment>
```

Header line is just name in square bracket:

```
[<section name>]
```

Data line contain information token then sign '=' and a string of bytes in C byte array format:

```
<token> = <C byte array>
```

**Tokens list:**

- `plaintext`

    Original plaintext to be encrypted.

- `ciphertext`

    Encrypted plaintext string.

- `cipher_key`

    Key used in cipher operation.

- `auth_key`

    Key used in auth operation.

- `cipher_iv`

    Cipher Initial Vector.

- `auth_iv`

    Auth Initial Vector.

- `aad`

    Additional data.

- digest

    Digest string.

## 6.4 Examples

Call application for performance throughput test of single Aesni MB PMD for cipher encryption aes-cbc and auth generation sha1-hmac, one million operations, burst size 32, packet size 64:

```
dpdk-test-crypto-perf -l 6-7 --vdev crypto_aesni_mb -w 0000:00:00.0 --
--ptest throughput --devtype crypto_aesni_mb --optype cipher-then-auth
--cipher-algo aes-cbc --cipher-op encrypt --cipher-key-sz 16 --auth-algo
sha1-hmac --auth-op generate --auth-key-sz 64 --digest-sz 12
--total-ops 10000000 --burst-sz 32 --buffer-sz 64
```

Call application for performance latency test of two Aesni MB PMD executed on two cores for cipher encryption aes-cbc, ten operations in silent mode:

```
dpdk-test-crypto-perf -l 4-7 --vdev crypto_aesni_mb1
--vdev crypto_aesni_mb2 -w 0000:00:00.0 -- --devtype crypto_aesni_mb
--cipher-algo aes-cbc --cipher-key-sz 16 --cipher-iv-sz 16
--cipher-op encrypt --optype cipher-only --silent
--ptest latency --total-ops 10
```

Call application for verification test of single open ssl PMD for cipher encryption aes-gcm and auth generation aes-gcm,ten operations in silent mode, test vector provide in file "test_aes_gcm.data" with packet verification:

```
dpdk-test-crypto-perf -l 4-7 --vdev crypto_openssl -w 0000:00:00.0 --
--devtype crypto_openssl --aead-algo aes-gcm --aead-key-sz 16
--aead-iv-sz 16 --aead-op encrypt --aead-aad-sz 16 --digest-sz 16
--optype aead --silent --ptest verify --total-ops 10
--test-file test_aes_gcm.data
```

Test vector file for cipher algorithm aes cbc 256 with authorization sha:

```
# Global Section
plaintext =
0xff, 0xca, 0xfb, 0xf1, 0x38, 0x20, 0x2f, 0x7b, 0x24, 0x98, 0x26, 0x7d, 0x1d, 0x9f, 0xb3, 0x93,
0xd9, 0xef, 0xbd, 0xad, 0x4e, 0x40, 0xbd, 0x60, 0xe9, 0x48, 0x59, 0x90, 0x67, 0xd7, 0x2b, 0x7b,
0x8a, 0xe0, 0x4d, 0xb0, 0x70, 0x38, 0xcc, 0x48, 0x61, 0x7d, 0xee, 0xd6, 0x35, 0x49, 0xae, 0xb4,
0xaf, 0x6b, 0xdd, 0xe6, 0x21, 0xc0, 0x60, 0xce, 0x0a, 0xf4, 0x1c, 0x2e, 0x1c, 0x8d, 0xe8, 0x7b
ciphertext =
0x77, 0xF9, 0xF7, 0x7A, 0xA3, 0xCB, 0x68, 0x1A, 0x11, 0x70, 0xD8, 0x7A, 0xB6, 0xE2, 0x37, 0x7E,
0xD1, 0x57, 0x1C, 0x8E, 0x85, 0xD8, 0x08, 0xBF, 0x57, 0x1F, 0x21, 0x6C, 0xAD, 0xAD, 0x47, 0x1E,
0x0D, 0x6B, 0x79, 0x39, 0x15, 0x4E, 0x5B, 0x59, 0x2D, 0x76, 0x87, 0xA6, 0xD6, 0x47, 0x8F, 0x82,
0xB8, 0x51, 0x91, 0x32, 0x60, 0xCB, 0x97, 0xDE, 0xBE, 0xF0, 0xAD, 0xFC, 0x23, 0x2E, 0x22, 0x02
cipher_key =
0xE4, 0x23, 0x33, 0x8A, 0x35, 0x64, 0x61, 0xE2, 0x49, 0x03, 0xDD, 0xC6, 0xB8, 0xCA, 0x55, 0x7A,
0xd0, 0xe7, 0x4b, 0xfb, 0x5d, 0xe5, 0x0c, 0xe7, 0x6f, 0x21, 0xb5, 0x52, 0x2a, 0xbb, 0xc7, 0xf7
auth_key =
0xaf, 0x96, 0x42, 0xf1, 0x8c, 0x50, 0xdc, 0x67, 0x1a, 0x43, 0x47, 0x62, 0xc7, 0x04, 0xab, 0x05,
0xf5, 0x0c, 0xe7, 0xa2, 0xa6, 0x23, 0xd5, 0x3d, 0x95, 0xd8, 0xcd, 0x86, 0x79, 0xf5, 0x01, 0x47,
0x4f, 0xf9, 0x1d, 0x9d, 0x36, 0xf7, 0x68, 0x1a, 0x64, 0x44, 0x58, 0x5d, 0xe5, 0x81, 0x15, 0x2a,
0x41, 0xe4, 0x0e, 0xaa, 0x1f, 0x04, 0x21, 0xff, 0x2c, 0xf3, 0x73, 0x2b, 0x48, 0x1e, 0xd2, 0xf7
cipher_iv =
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F
# Section sha 1 hmac buff 32
[sha1_hmac_buff_32]
digest =
0x36, 0xCA, 0x49, 0x6A, 0xE3, 0x54, 0xD8, 0x4F, 0x0B, 0x76, 0xD8, 0xAA, 0x78, 0xEB, 0x9D, 0x65,
0x2C, 0xCA, 0x1F, 0x97
```

```
# Section sha 256 hmac buff 32
[sha256_hmac_buff_32]
digest =
0x1C, 0xB2, 0x3D, 0xD1, 0xF9, 0xC7, 0x6C, 0x49, 0x2E, 0xDA, 0x94, 0x8B, 0xF1, 0xCF, 0x96, 0x43,
0x67, 0x50, 0x39, 0x76, 0xB5, 0xA1, 0xCE, 0xA1, 0xD7, 0x77, 0x10, 0x07, 0x43, 0x37, 0x05, 0xB4
```

# DPDK-TEST-COMPRESS-PERF TOOL

The `dpdk-test-compress-perf` tool is a Data Plane Development Kit (DPDK) utility that allows measuring performance parameters of PMDs available in the compress tree. User can use multiple cores to run tests on but only one type of compression PMD can be measured during single application execution. The tool reads the data from a file (–input-file), dumps all the file into a buffer and fills out the data of input mbufs, which are passed to compress device with compression operations. Then, the output buffers are fed into the decompression stage, and the resulting data is compared against the original data (verification phase). After that, a number of iterations are performed, compressing first and decompressing later, to check the throughput rate (showing cycles/iteration, cycles/Byte and Gbps, for compression and decompression).

**Note:** if the max-num-sgl-segs x seg_sz > input size then segments number in the chain will be lower than value passed into max-num-sgl-segs.

## 7.1 Limitations

- Stateful operation is not supported in this version.

## 7.2 EAL Options

The following are the EAL command-line options that can be used in conjunction with the `dpdk-test-compress-perf` application. See the DPDK Getting Started Guides for more information on these options.

- `-c <COREMASK>` or `-l <CORELIST>`

    Set the hexadecimal bitmask of the cores to run on. The corelist is a list cores to use.

**Note:** One lcore is needed for process admin, tests are run on all other cores. To run tests on two lcores, three lcores must be passed to the tool.

- `-w <PCI>`

    Add a PCI device in white list.

- `--vdev <driver><id>`

    Add a virtual device.

## 7.3 Application Options

`--ptest [benchmark/verify]`: set test type (default: benchmark)

`--driver-name NAME`: compress driver to use

`--input-file NAME`: file to compress and decompress

`--extended-input-sz N`: extend file data up to this size (default: no extension)

`--seg-sz N`: size of segment to store the data (default: 2048)

`--burst-sz N`: compress operation burst size

`--pool-sz N`: mempool size for compress operations/mbufs (default: 8192)

`--max-num-sgl-segs N`: maximum number of segments for each mbuf (default: 16)

`--num-iter N`: number of times the file will be compressed/decompressed (default: 10000)

`--operation [comp/decomp/comp_and_decomp]`: perform test on compression, decompression or both operations

`--huffman-enc [fixed/dynamic/default]`: Huffman encoding (default: dynamic)

`--compress-level N`: compression level, which could be a single value, list or range (default: range between 1 and 9)

`--window-sz N`: base two log value of compression window size (default: max supported by PMD)

`-h`: prints this help

### 7.3.1 Compiling the Tool

**Step 1: PMD setting**

The `dpdk-test-compress-perf` tool depends on compression device drivers PMD which can be disabled by default in the build configuration file `common_base`. The compression device drivers PMD which should be tested can be enabled by setting e.g.:

```
CONFIG_RTE_LIBRTE_PMD_ISAL=y
```

### 7.3.2 Running the Tool

The tool has a number of command line options. Here is the sample command line:

```
./build/app/dpdk-test-compress-perf  -l 4 -- --driver-name compress_qat --input-file test.txt
  --compress-level 1:1:9 --num-iter 10 --extended-input-sz 1048576  --max-num-sgl-segs 16 --huff
```

# DPDK-TEST-EVENTDEV APPLICATION

The `dpdk-test-eventdev` tool is a Data Plane Development Kit (DPDK) application that allows exercising various eventdev use cases. This application has a generic framework to add new eventdev based test cases to verify functionality and measure the performance parameters of DPDK eventdev devices.

## 8.1 Compiling the Application

**Build the application**

Execute the `dpdk-setup.sh` script to build the DPDK library together with the `dpdk-test-eventdev` application.

Initially, the user must select a DPDK target to choose the correct target type and compiler options to use when building the libraries. The user must have all libraries, modules, updates and compilers installed in the system prior to this, as described in the earlier chapters in this Getting Started Guide.

## 8.2 Running the Application

The application has a number of command line options:

```
dpdk-test-eventdev [EAL Options] -- [application options]
```

### 8.2.1 EAL Options

The following are the EAL command-line options that can be used in conjunction with the `dpdk-test-eventdev` application. See the DPDK Getting Started Guides for more information on these options.

- `-c <COREMASK>` or `-l <CORELIST>`

    Set the hexadecimal bitmask of the cores to run on. The corelist is a list of cores to use.

- `--vdev <driver><id>`

    Add a virtual eventdev device.

## 8.2.2 Application Options

The following are the application command-line options:

- `--verbose`

  Set verbose level. Default is 1. Value > 1 displays more details.

- `--dev <n>`

  Set the device id of the event device.

- `--test <name>`

  Set test name, where `name` is one of the following:

  ```
  order_queue
  order_atq
  perf_queue
  perf_atq
  pipeline_atq
  pipeline_queue
  ```

- `--socket_id <n>`

  Set the socket id of the application resources.

- `--pool-sz <n>`

  Set the number of mbufs to be allocated from the mempool.

- `--plcores <CORELIST>`

  Set the list of cores to be used as producers.

- `--wlcores <CORELIST>`

  Set the list of cores to be used as workers.

- `--stlist <type_list>`

  Set the scheduled type of each stage where `type_list` size determines the number of stages used in the test application. Each type_list member can be one of the following:

  ```
  P or p : Parallel schedule type
  O or o : Ordered schedule type
  A or a : Atomic schedule type
  ```

  Application expects the `type_list` in comma separated form (i.e. `--stlist o,a,a,a`)

- `--nb_flows <n>`

  Set the number of flows to produce.

- `--nb_pkts <n>`

  Set the number of packets to produce. 0 implies no limit.

- `--worker_deq_depth <n>`

  Set the dequeue depth of the worker.

- `--fwd_latency`

  Perform forward latency measurement.

- `--queue_priority`

    Enable queue priority.

- `--prod_type_ethdev`

    Use ethernet device as producer.

- `--prod_type_timerdev`

    Use event timer adapter as producer.

- `--prod_type_timerdev_burst`

    Use burst mode event timer adapter as producer.

- `--timer_tick_nsec`

    Used to dictate number of nano seconds between bucket traversal of the event timer adapter. Refer *rte_event_timer_adapter_conf*.

- `--max_tmo_nsec`

    Used to configure event timer adapter max arm timeout in nano seconds.

- `--expiry_nsec`

    Dictate the number of nano seconds after which the event timer expires.

- `--nb_timers`

    Number of event timers each producer core will generate.

- `--nb_timer_adptrs`

    Number of event timer adapters to be used. Each adapter is used in round robin manner by the producer cores.

- `--deq_tmo_nsec`

    Global dequeue timeout for all the event ports if the provided dequeue timeout is out of the supported range of event device it will be adjusted to the highest/lowest supported dequeue timeout supported.

## 8.3 Eventdev Tests

### 8.3.1 ORDER_QUEUE Test

This is a functional test case that aims at testing the following:

1. Verify the ingress order maintenance.

2. Verify the exclusive(atomic) access to given atomic flow per eventdev port.

Table 8.1: Order queue test eventdev configuration.

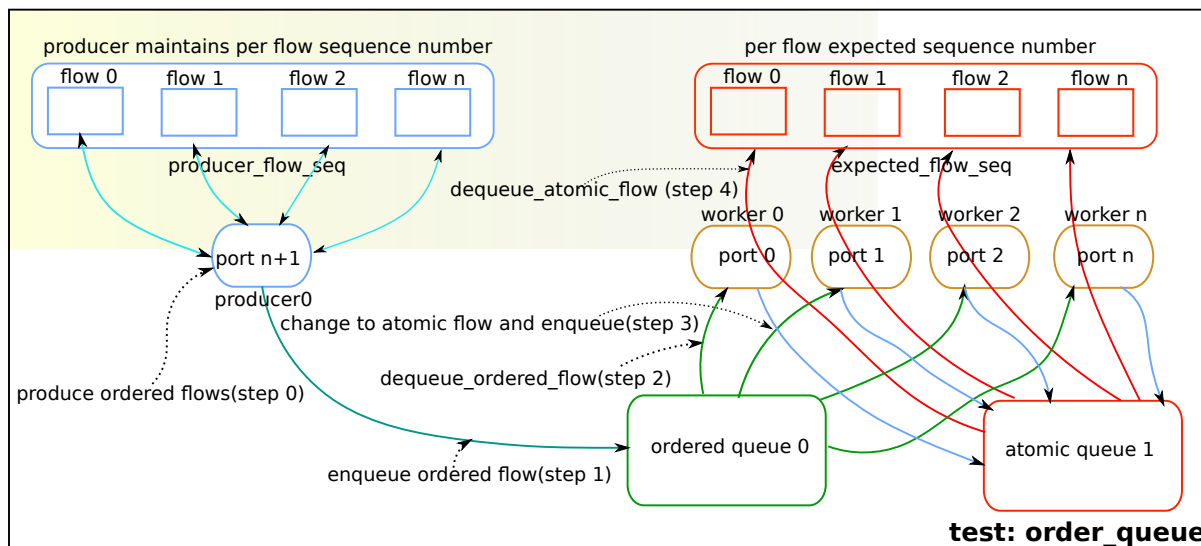| # | Items | Value | Comments |
|---|-------|-------|----------|
| 1 | nb_queues | 2 | q0(ordered), q1(atomic) |
| 2 | nb_producers | 1 | |
| 3 | nb_workers | >= 1 | |
| 4 | nb_ports | nb_workers + 1 | Workers use port 0 to port n-1. Producer uses port n |



Fig. 8.1: order queue test operation.

The order queue test configures the eventdev with two queues and an event producer to inject the events to q0(ordered) queue. Both q0(ordered) and q1(atomic) are linked to all the workers.

The event producer maintains a sequence number per flow and injects the events to the ordered queue. The worker receives the events from ordered queue and forwards to atomic queue. Since the events from an ordered queue can be processed in parallel on the different workers, the ingress order of events might have changed on the downstream atomic queue enqueue. On enqueue to the atomic queue, the eventdev PMD driver reorders the event to the original ingress order(i.e producer ingress order).

When the event is dequeued from the atomic queue by the worker, this test verifies the expected sequence number of associated event per flow by comparing the free running expected sequence number per flow.

**Application options**

Supported application command line options are following:

```
--verbose
--dev
--test
--socket_id
--pool_sz
--plcores
--wlcores
--nb_flows
--nb_pkts
--worker_deq_depth
--deq_tmo_nsec
```

**Example**

Example command to run order queue test:

```
sudo build/app/dpdk-test-eventdev --vdev=event_sw0 -- \
          --test=order_queue --plcores 1 --wlcores 2,3
```

## 8.3.2 ORDER_ATQ Test

This test verifies the same aspects of `order_queue` test, the difference is the number of queues used, this test operates on a single `all types queue(atq)` instead of two different queues for ordered and atomic.

Table 8.2: Order all types queue test eventdev configuration.

| # | Items | Value | Comments |
|---|-------|-------|----------|
| 1 | nb_queues | 1 | q0(all types queue) |
| 2 | nb_producers | 1 | |
| 3 | nb_workers | >= 1 | |
| 4 | nb_ports | nb_workers + 1 | Workers use port 0 to port n-1.Producer uses port n. |
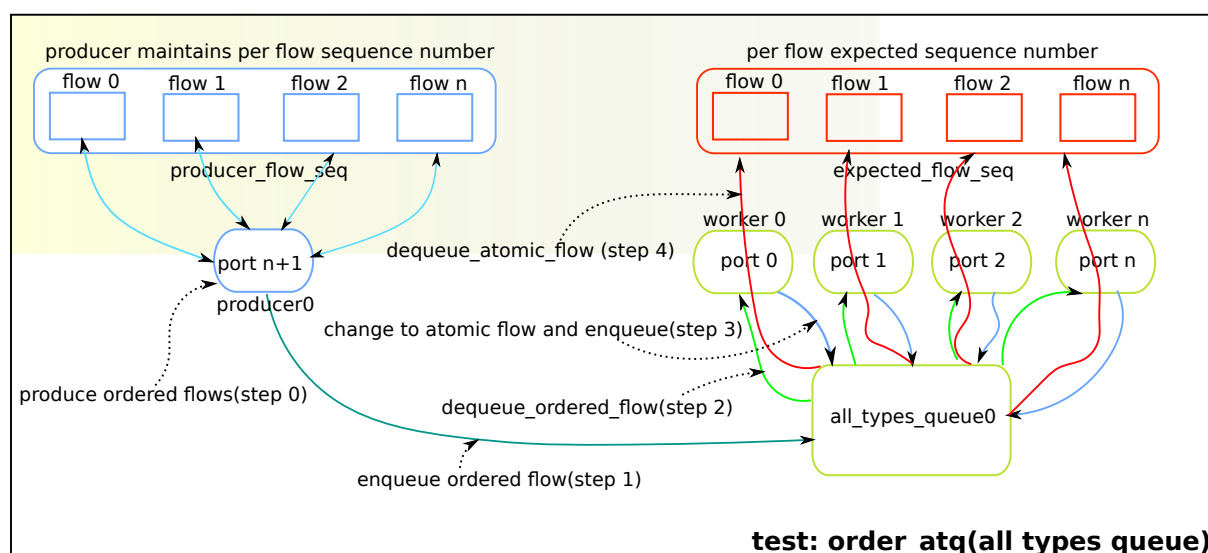


Fig. 8.2: order all types queue test operation.

**Application options**

Supported application command line options are following:

```
--verbose
--dev
--test
--socket_id
--pool_sz
--plcores
--wlcores
--nb_flows
--nb_pkts
```

```
--worker_deq_depth
--deq_tmo_nsec
```

**Example**

Example command to run order `all types queue` test:

```
sudo build/app/dpdk-test-eventdev --vdev=event_octeontx -- \
                --test=order_atq --plcores 1 --wlcores 2,3
```

### 8.3.3 PERF_QUEUE Test

This is a performance test case that aims at testing the following:

1. Measure the number of events can be processed in a second.

2. Measure the latency to forward an event.

Table 8.3: Perf queue test eventdev configuration.

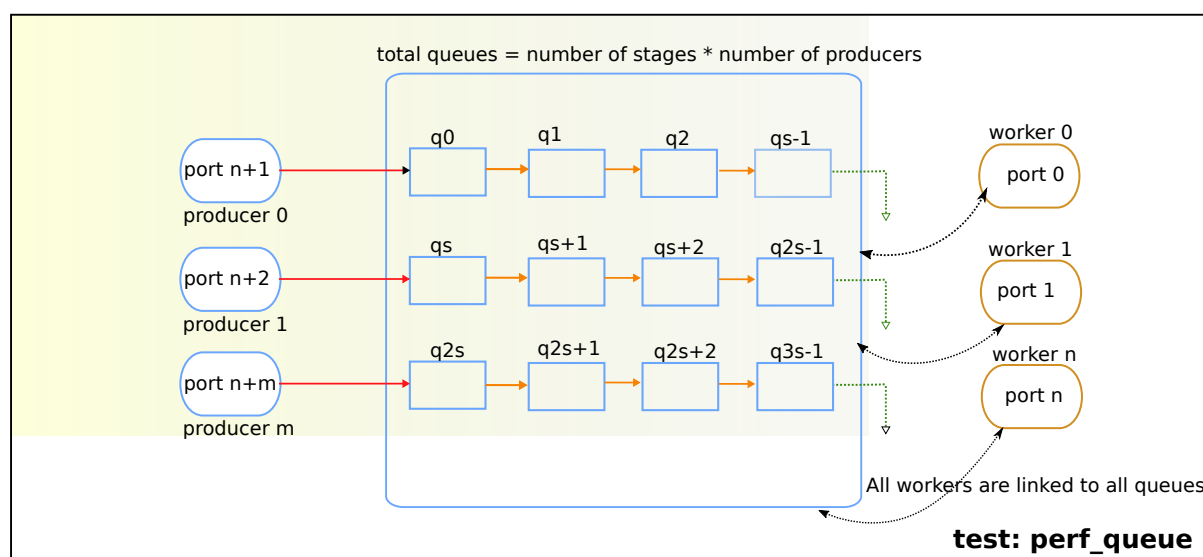| # | Items | Value | Comments |
|---|-------|-------|----------|
| 1 | nb_queues | nb_producers * nb_stages | Queues will be configured based on the user requested sched type list(–stlist) |
| 2 | nb_producers | >= 1 | Selected through –plcores command line argument. |
| 3 | nb_workers | >= 1 | Selected through –wlcores command line argument |
| 4 | nb_ports | nb_workers + nb_producers | Workers use port 0 to port n-1. Producers use port n to port p |



Fig. 8.3: perf queue test operation.

The perf queue test configures the eventdev with Q queues and P ports, where Q and P is a function of the number of workers, the number of producers and number of stages as mentioned in Table 8.3.

The user can choose the number of workers, the number of producers and number of stages through the `--wlcores`, `--plcores` and the `--stlist` application command line arguments respectively.

The producer(s) injects the events to eventdev based the first stage sched type list requested by the user through `--stlist` the command line argument.

Based on the number of stages to process(selected through `--stlist`), The application forwards the event to next upstream queue and terminates when it reaches the last stage in the pipeline. On event termination, application increments the number events processed and print periodically in one second to get the number of events processed in one second.

When `--fwd_latency` command line option selected, the application inserts the timestamp in the event on the first stage and then on termination, it updates the number of cycles to forward a packet. The application uses this value to compute the average latency to a forward packet.

When `--prod_type_ethdev` command line option is selected, the application uses the probed ethernet devices as producers by configuring them as Rx adapters instead of using synthetic producers.

### Application options

Supported application command line options are following:

```
--verbose
--dev
--test
--socket_id
--pool_sz
--plcores
--wlcores
--stlist
--nb_flows
--nb_pkts
--worker_deq_depth
--fwd_latency
--queue_priority
--prod_type_ethdev
--prod_type_timerdev_burst
--prod_type_timerdev
--timer_tick_nsec
--max_tmo_nsec
--expiry_nsec
--nb_timers
--nb_timer_adptrs
--deq_tmo_nsec
```

### Example

Example command to run perf queue test:

```
sudo build/app/dpdk-test-eventdev -c 0xf -s 0x1 --vdev=event_sw0 -- \
    --test=perf_queue --plcores=2 --wlcore=3 --stlist=p --nb_pkts=0
```

Example command to run perf queue test with ethernet ports:

```
sudo build/app/dpdk-test-eventdev --vdev=event_sw0 -- \
    --test=perf_queue --plcores=2 --wlcore=3 --stlist=p --prod_type_ethdev
```

Example command to run perf queue test with event timer adapter:

```
sudo  build/app/dpdk-test-eventdev --vdev="event_octeontx" -- \
         --wlcores 4 --plcores 12 --test perf_queue --stlist=a \
         --prod_type_timerdev --fwd_latency
```

### 8.3.4 PERF_ATQ Test

This is a performance test case that aims at testing the following with `all types queue` eventdev scheme.

1. Measure the number of events can be processed in a second.

2. Measure the latency to forward an event.

Table  8.4: Perf all types queue test eventdev configuration.

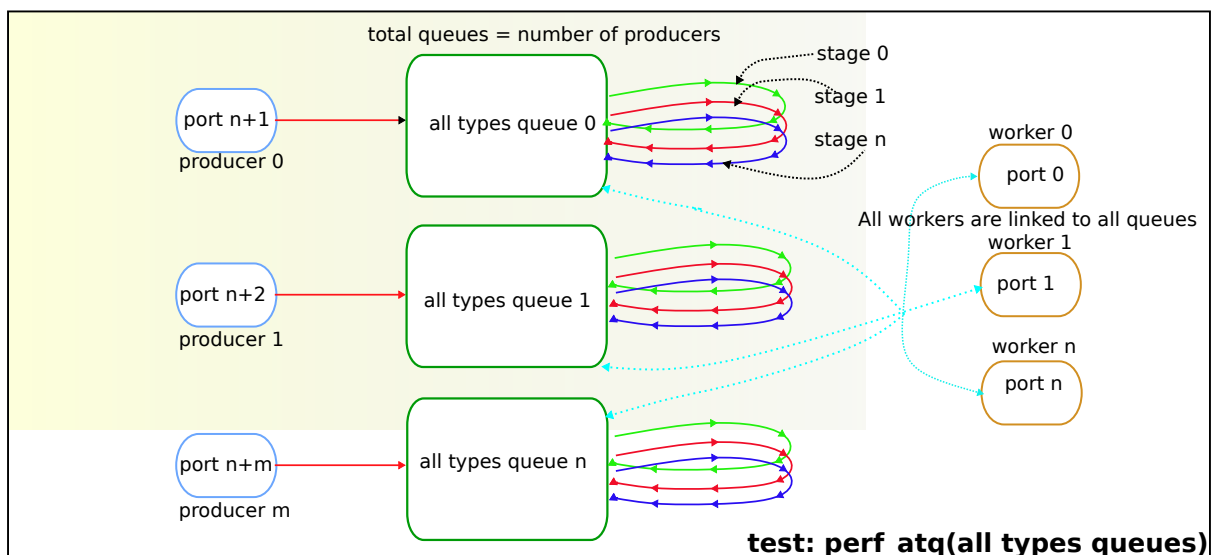| # | Items | Value | Comments |
|---|---|---|---|
| 1 | nb_queues | nb_producers | Queues will be configured based on the user requested sched type list(–stlist) |
| 2 | nb_producers | >= 1 | Selected through –plcores command line argument. |
| 3 | nb_workers | >= 1 | Selected through –wlcores command line argument |
| 4 | nb_ports | nb_workers + nb_producers | Workers use port 0 to port n-1. Producers use port n to port p |



Fig.  8.4: perf all types queue test operation.

The `all types queues(atq)` perf test configures the eventdev with Q queues and P ports, where Q and P is a function of the number of workers and number of producers as mentioned in Table 8.4.

The atq queue test functions as same as `perf_queue` test. The difference is, It uses, `all type queue scheme` instead of separate queues for each stage and thus reduces the number of queues required to realize the use case and enables flow pinning as the event does not move to the next queue.

### Application options

Supported application command line options are following:

```
--verbose
--dev
--test
--socket_id
--pool_sz
--plcores
--wlcores
--stlist
--nb_flows
--nb_pkts
--worker_deq_depth
--fwd_latency
--prod_type_ethdev
--prod_type_timerdev_burst
--prod_type_timerdev
--timer_tick_nsec
--max_tmo_nsec
--expiry_nsec
--nb_timers
--nb_timer_adptrs
--deq_tmo_nsec
```

### Example

Example command to run perf `all types queue` test:

```
sudo build/app/dpdk-test-eventdev --vdev=event_octeontx -- \
        --test=perf_atq --plcores=2 --wlcore=3 --stlist=p --nb_pkts=0
```

Example command to run perf `all types queue` test with event timer adapter:

```
sudo  build/app/dpdk-test-eventdev --vdev="event_octeontx" -- \
        --wlcores 4 --plcores 12 --test perf_atq --verbose 20 \
        --stlist=a --prod_type_timerdev --fwd_latency
```

### 8.3.5 PIPELINE_QUEUE Test

This is a pipeline test case that aims at testing the following:

1. Measure the end-to-end performance of an event dev with a ethernet dev.

2. Maintain packet ordering from Rx to Tx.

Table 8.5: Pipeline queue test eventdev configuration.

| # | Items | Value | Comments |
|---|-------|-------|----------|
| 1 | nb_queues | (nb_producers * nb_stages) + nb_producers | Queues will be configured based on the user requested sched type list(–stlist) At the last stage of the schedule list the event is enqueued onto per port unique queue which is then Transmitted. |
| 2 | nb_producers | nb_eth | Producers will be configured based on the number of detected ethernet devices. Each ethdev will be configured as an Rx adapter. |
| 3 | nb_workers | wlcores | Selected through –wlcores command line argument |
| 4 | nb_ports | nb_workers + (nb_produces * 2) | Workers use port 0 to port n. Producers use port n+1 to port n+m, depending on the Rx adapter capability. Consumers use port n+m+1 to port n+o depending on the Tx adapter capability. |
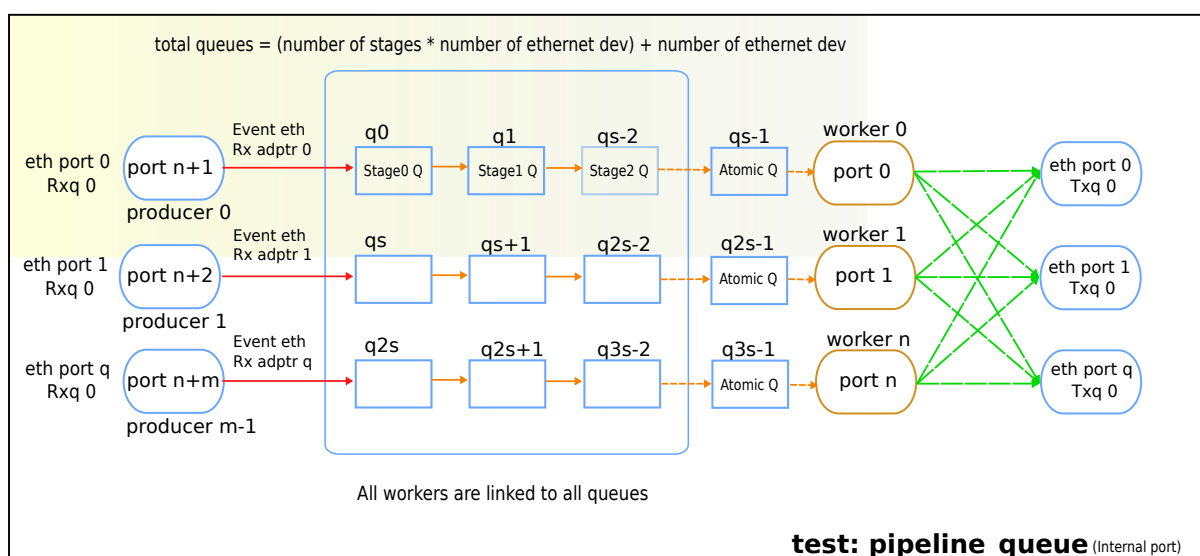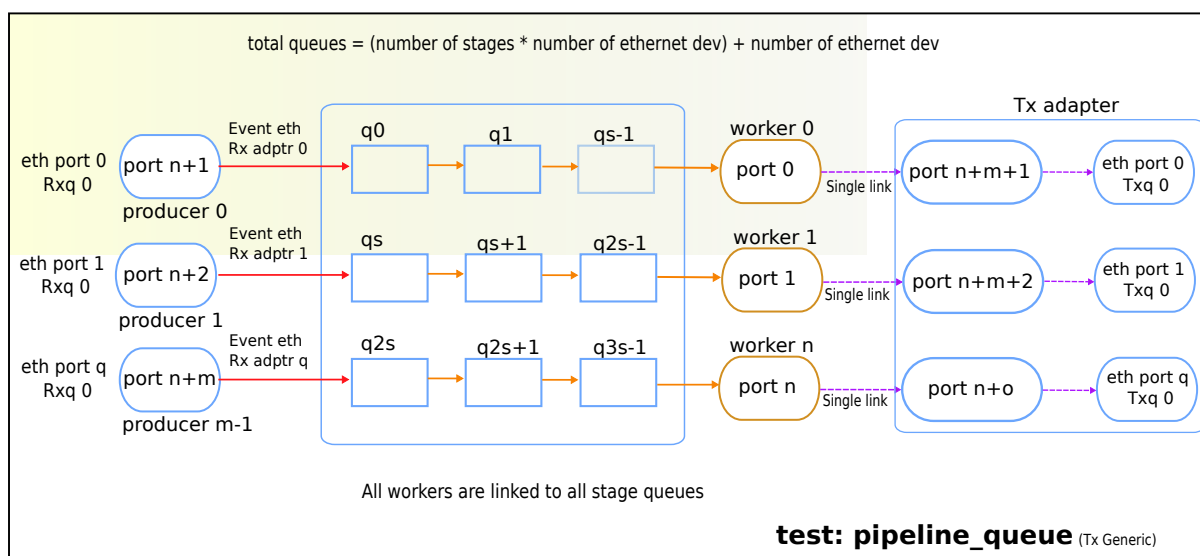




Fig. 8.5: pipeline queue test operation.

The pipeline queue test configures the eventdev with Q queues and P ports, where Q and P is a function of the number of workers, the number of producers and number of stages as mentioned in Table 8.5.

The user can choose the number of workers and number of stages through the `--wlcores` and the `--stlist` application command line arguments respectively.

The number of producers depends on the number of ethernet devices detected and each ethernet device is configured as a event_eth_rx_adapter that acts as a producer.

The producer(s) injects the events to eventdev based the first stage sched type list requested by the user through `--stlist` the command line argument.

Based on the number of stages to process(selected through `--stlist`), The application forwards the event to next upstream queue and when it reaches the last stage in the pipeline if the event type is `atomic` it is enqueued onto ethdev Tx queue else to maintain ordering the event type is set to `atomic` and enqueued onto the last stage queue.

If the ethdev and eventdev pair have `RTE_EVENT_ETH_TX_ADAPTER_CAP_INTERNAL_PORT` capability then the worker cores enqueue the packets to the eventdev directly using `rte_event_eth_tx_adapter_enqueue` else the worker cores enqueue the packet onto the `SINGLE_LINK_QUEUE` that is managed by the Tx adapter. The Tx adapter dequeues the packet and transmits it.

On packet Tx, application increments the number events processed and print periodically in one second to get the number of events processed in one second.

### Application options

Supported application command line options are following:

```
--verbose
--dev
--test
--socket_id
--pool_sz
--wlcores
--stlist
--worker_deq_depth
--prod_type_ethdev
--deq_tmo_nsec
```

**Note:**

- The `--prod_type_ethdev` is mandatory for running this test.

### Example

Example command to run pipeline queue test:

```
sudo build/app/dpdk-test-eventdev -c 0xf -s 0x8 --vdev=event_sw0 -- \
    --test=pipeline_queue --wlcore=1 --prod_type_ethdev --stlist=a
```
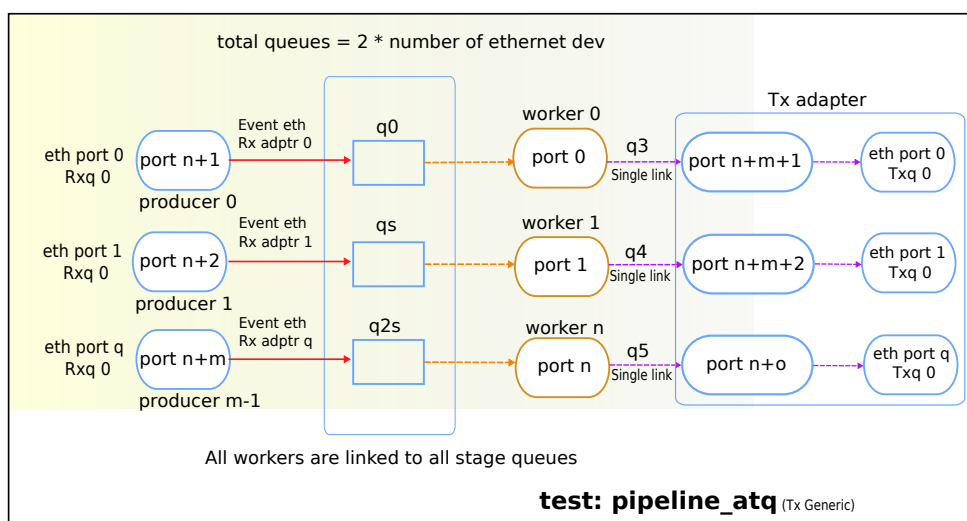
### 8.3.6 PIPELINE_ATQ Test

This is a pipeline test case that aims at testing the following with `all types queue` eventdev scheme.

1. Measure the end-to-end performance of an event dev with a ethernet dev.

2. Maintain packet ordering from Rx to Tx.

Table 8.6: Pipeline atq test eventdev configuration.

| # | Items | Value | Comments |
|---|---|---|---|
| 1 | nb_queues | nb_producers + x | Queues will be configured based on the user requested sched type list(–stlist) where x = nb_producers in generic pipeline and 0 if all the ethdev being used have Internal port capability |
| 2 | nb_producers | x | Producers will be configured based on the number of detected ethernet devices. Each ethdev will be configured as an Rx adapter. |
| 3 | nb_workers | x + 1 | Selected through –wlcores command line argument |
| 4 | nb_ports | nb_workers + nb_producers + x | Workers use port 0 to port n. Producers use port n+1 to port n+m, depending on the Rx adapter capability. x = nb_producers in generic pipeline and 0 if all the ethdev being used have Internal port capability. Consumers may use port n+m+1 to port n+o depending on the Tx adapter capability. |



The pipeline atq test configures the eventdev with Q queues and P ports, where Q and P is a function of the number of workers, the number of producers and number of stages as mentioned in Table 8.6.

The atq queue test functions as same as `pipeline_queue` test. The difference is, It uses, `all type queue scheme` instead of separate queues for each stage and thus reduces the number of queues required to realize the use case.

### Application options

Supported application command line options are following:

```
--verbose
--dev
```

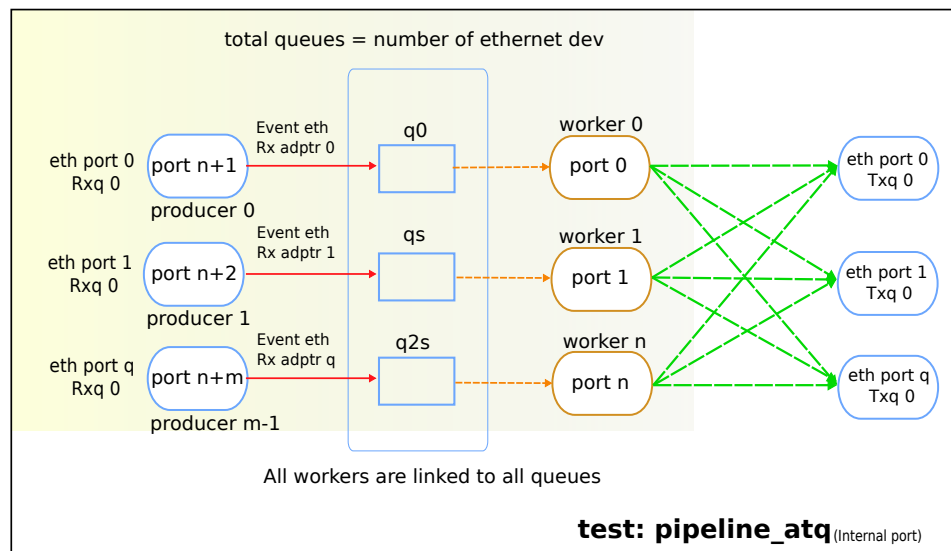Fig. 8.6: pipeline atq test operation.

```
--test
--socket_id
--pool_sz
--wlcores
--stlist
--worker_deq_depth
--prod_type_ethdev
--deq_tmo_nsec
```

**Note:**

- The `--prod_type_ethdev` is mandatory for running this test.

## Example

Example command to run pipeline queue test:

```
sudo build/app/dpdk-test-eventdev -c 0xf -s 0x8 --vdev=event_sw0 -- \
    --test=pipeline_atq --wlcore=1 --prod_type_ethdev --stlist=a
```