DPDK

DATA PLANE DEVELOPMENT KIT

Baseband Device Drivers

Release 19.08.2

Nov 15, 2019

CHAPTER

ONE

BBDEV NULL POLL MODE DRIVER

The (**baseband_null**) is a bbdev poll mode driver which provides a minimal implementation of a software bbdev device. As a null device it does not modify the data in the mbuf on which the bbdev operation is to operate and it only works for operation type `RTE_BBDEV_OP_NONE`.

When a burst of mbufs is submitted to a *bbdev null PMD* for processing then each mbuf in the burst will be enqueued in an internal buffer ring to be collected on a dequeue call.

1.1 Limitations

- In-place operations for Turbo encode and decode are not supported

1.2 Installation

The *bbdev null PMD* is enabled and built by default in both the Linux and FreeBSD builds.

1.3 Initialization

To use the PMD in an application, user must:

- Call `rte_vdev_init("baseband_null")` within the application.
- Use `--vdev="baseband_null"` in the EAL options, which will call `rte_vdev_init()` internally.

The following parameters (all optional) can be provided in the previous two calls:

- `socket_id`: Specify the socket where the memory for the device is going to be allocated (by default, *socket_id* will be the socket where the core that is creating the PMD is running on).
- `max_nb_queues`: Specify the maximum number of queues in the device (default is `RTE_MAX_LCORE`).

1.3.1 Example:

```
./test-bbdev.py -e="--vdev=baseband_null,socket_id=0,max_nb_queues=8"
```

CHAPTER

TWO

SW TURBO POLL MODE DRIVER

The SW Turbo PMD (**baseband_turbo_sw**) provides a software only poll mode bbdev driver that can optionally utilize Intel optimized libraries for LTE and 5GNR Layer 1 workloads acceleration.

Note that the driver can also be built without any dependency with reduced functionality for maintenance purpose.

To enable linking to the SDK libraries see detailed installation section below. Two flags can be enabled depending on whether the target machine can support AVX2 and AVX512 instructions sets and the related SDK libraries for vectorized signal processing functions are installed : - CONFIG_RTE_BBDEV_SDK_AVX2 - CONFIG_RTE_BBDEV_SDK_AVX512 By default these 2 flags are disabled by default. For AVX2 machine and SDK library installed then the first flag can be enabled. For AVX512 machine and SDK library installed then both flags can be enabled for full real time capability.

This PMD supports the functions: FEC, Rate Matching and CRC functions detailed in the Features section.

2.1 Features

SW Turbo PMD can support for the following capabilities when the SDK libraries are used:

For the LTE encode operation:

- `RTE_BBDEV_TURBO_CRC_24A_ATTACH`
- `RTE_BBDEV_TURBO_CRC_24B_ATTACH`
- `RTE_BBDEV_TURBO_RATE_MATCH`
- `RTE_BBDEV_TURBO_RV_INDEX_BYPASS`

For the LTE decode operation:

- `RTE_BBDEV_TURBO_SUBBLOCK_DEINTERLEAVE`
- `RTE_BBDEV_TURBO_CRC_TYPE_24B`
- `RTE_BBDEV_TURBO_POS_LLR_1_BIT_IN`
- `RTE_BBDEV_TURBO_NEG_LLR_1_BIT_IN`
- `RTE_BBDEV_TURBO_DEC_TB_CRC_24B_KEEP`
- `RTE_BBDEV_TURBO_EARLY_TERMINATION`

For the 5G NR LDPC encode operation:

- RTE_BBDEV_LDPC_RATE_MATCH
- RTE_BBDEV_LDPC_CRC_24A_ATTACH
- RTE_BBDEV_LDPC_CRC_24B_ATTACH

For the 5G NR LDPC decode operation:

- RTE_BBDEV_LDPC_CRC_TYPE_24B_CHECK
- RTE_BBDEV_LDPC_CRC_TYPE_24A_CHECK
- RTE_BBDEV_LDPC_CRC_TYPE_24B_DROP
- RTE_BBDEV_LDPC_HQ_COMBINE_IN_ENABLE
- RTE_BBDEV_LDPC_HQ_COMBINE_OUT_ENABLE
- RTE_BBDEV_LDPC_ITERATION_STOP_ENABLE

2.2 Limitations

- In-place operations for encode and decode are not supported

2.3 Installation

2.3.1 FlexRAN SDK Download

As an option it is possible to link this driver with FleXRAN SDK libraries which can enable real time signal processing using AVX instructions.

These libraries are available through this link.

After download is complete, the user needs to unpack and compile on their system before building DPDK.

The following table maps DPDK versions with past FlexRAN SDK releases:

Table 2.1: DPDK and FlexRAN FEC SDK releases compliance

DPDK version	FlexRAN FEC SDK release
19.08	19.04

2.3.2 FlexRAN SDK Installation

Note that the installation of these libraries is optional.

The following are pre-requisites for building FlexRAN SDK Libraries:

1. An AVX2 or AVX512 supporting machine
2. CentOS Linux release 7.2.1511 (Core) operating system is advised
3. Intel ICC 18.0.1 20171018 compiler or more recent and related libraries ICC is available with a free community license.

The following instructions should be followed in this exact order:

1. Set the environment variables:

```
source <path-to-icc-compiler-install-folder>/linux/bin/compilervars.sh intel64 -platf
```

2. Run the SDK extractor script and accept the license:

```
cd <path-to-workspace>
./FlexRAN-FEC-SDK-19-04.sh
```

3. Generate makefiles based on system configuration:

```
cd <path-to-workspace>/FlexRAN-FEC-SDK-19-04/sdk/
./create-makefiles-linux.sh
```

4. A build folder is generated in this form `build-<ISA>-<CC>`, enter that folder and install:

```
cd build-avx512-icc/
make && make install
```

2.4 Initialization

In order to enable this virtual bbdev PMD, the user may:

- Build the `FLEXRAN SDK` libraries (explained in Installation section).
- Export the environmental variables `FLEXRAN_SDK` to the path where the FlexRAN SDK libraries were installed. And `DIR_WIRELESS_SDK` to the path where the libraries were extracted.

Example:

```
export FLEXRAN_SDK=<path-to-workspace>/FlexRAN-FEC-SDK-19-04/sdk/build-avx2-icc/install
export DIR_WIRELESS_SDK=<path-to-workspace>/FlexRAN-FEC-SDK-19-04/sdk/build-avx2-icc/
```

- Set `CONFIG_RTE_BBDEV_SDK_AVX2=y` and `CONFIG_RTE_BBDEV_SDK_AVX512=y` in DPDK common configuration file `config/common_base` to be able to use the SDK libraries as mentioned above. For AVX2 machine it is possible to only enable CONFIG_RTE_BBDEV_SDK_AVX2 for limited 4G functionality. If no flag are set the PMD driver will still build but its capabilities will be limited accordingly.

To use the PMD in an application, user must:

- Call `rte_vdev_init("baseband_turbo_sw")` within the application.
- Use `--vdev="baseband_turbo_sw"` in the EAL options, which will call `rte_vdev_init()` internally.

The following parameters (all optional) can be provided in the previous two calls:

- `socket_id`: Specify the socket where the memory for the device is going to be allocated (by default, *socket_id* will be the socket where the core that is creating the PMD is running on).
- `max_nb_queues`: Specify the maximum number of queues in the device (default is `RTE_MAX_LCORE`).

2.4.1 Example:

```
./test-bbdev.py -e="--vdev=baseband_turbo_sw,socket_id=0,max_nb_queues=8" \
-c validation -v ./turbo_*_default.data
```

CHAPTER

THREE

INTEL(R) FPGA LTE FEC POLL MODE DRIVER

The BBDEV FPGA LTE FEC poll mode driver (PMD) supports an FPGA implementation of a VRAN Turbo Encode / Decode LTE wireless acceleration function, using Intel's PCI-e and FPGA based Vista Creek device.

3.1 Features

FPGA LTE FEC PMD supports the following features:

- Turbo Encode in the DL with total throughput of 4.5 Gbits/s
- Turbo Decode in the UL with total throughput of 1.5 Gbits/s assuming 8 decoder iterations
- 8 VFs per PF (physical device)
- Maximum of 32 UL queues per VF
- Maximum of 32 DL queues per VF
- PCIe Gen-3 x8 Interface
- MSI-X
- SR-IOV

FPGA LTE FEC PMD supports the following BBDEV capabilities:

- **For the turbo encode operation:**
  - `RTE_BBDEV_TURBO_CRC_24B_ATTACH` : set to attach CRC24B to CB(s)
  - `RTE_BBDEV_TURBO_RATE_MATCH` : if set then do not do Rate Match bypass
  - `RTE_BBDEV_TURBO_ENC_INTERRUPTS` : set for encoder dequeue interrupts
- **For the turbo decode operation:**
  - `RTE_BBDEV_TURBO_CRC_TYPE_24B` : check CRC24B from CB(s)
  - `RTE_BBDEV_TURBO_SUBBLOCK_DEINTERLEAVE` : perform subblock de-interleave
  - `RTE_BBDEV_TURBO_DEC_INTERRUPTS` : set for decoder dequeue interrupts
  - `RTE_BBDEV_TURBO_NEG_LLR_1_BIT_IN` : set if negative LLR encoder i/p is supported

  - RTE_BBDEV_TURBO_DEC_TB_CRC_24B_KEEP : keep CRC24B bits appended while decoding

3.2 Limitations

FPGA LTE FEC does not support the following:

- Scatter-Gather function

3.3 Installation

Section 3 of the DPDK manual provides instuctions on installing and compiling DPDK. The default set of bbdev compile flags may be found in config/common_base, where for example the flag to build the FPGA LTE FEC device, CONFIG_RTE_LIBRTE_PMD_FPGA_LTE_FEC, is already set. It is assumed DPDK has been compiled using for instance:

```
make install T=x86_64-native-linuxapp-gcc
```

DPDK requires hugepages to be configured as detailed in section 2 of the DPDK manual. The bbdev test application has been tested with a configuration 40 x 1GB hugepages. The hugepage configuration of a server may be examined using:

```
grep Huge* /proc/meminfo
```

3.4 Initialization

When the device first powers up, its PCI Physical Functions (PF) can be listed through this command:

```
sudo lspci -vd1172:5052
```

The physical and virtual functions are compatible with Linux UIO drivers: vfio and igb_uio. However, in order to work the FPGA LTE FEC device firstly needs to be bound to one of these linux drivers through DPDK.

3.4.1 Bind PF UIO driver(s)

Install the DPDK igb_uio driver, bind it with the PF PCI device ID and use lspci to confirm the PF device is under use by igb_uio DPDK UIO driver.

The igb_uio driver may be bound to the PF PCI device using one of three methods:

1. PCI functions (physical or virtual, depending on the use case) can be bound to the UIO driver by repeating this command for every function.

```
cd <dpdk-top-level-directory>
insmod ./build/kmod/igb_uio.ko
echo "1172 5052" > /sys/bus/pci/drivers/igb_uio/new_id
lspci -vd1172:
```

2. Another way to bind PF with DPDK UIO driver is by using the dpdk-devbind.py tool

```
cd <dpdk-top-level-directory>
./usertools/dpdk-devbind.py -b igb_uio 0000:06:00.0
```

where the PCI device ID (example: 0000:06:00.0) is obtained using lspci -vd1172:

3. A third way to bind is to use `dpdk-setup.sh` tool

```
cd <dpdk-top-level-directory>
./usertools/dpdk-setup.sh

select 'Bind Ethernet/Crypto/Baseband device to IGB UIO module'
or
select 'Bind Ethernet/Crypto/Baseband device to VFIO module' depending on driver required
enter PCI device ID
select 'Display current Ethernet/Crypto/Baseband device settings' to confirm binding
```

In the same way the FPGA LTE FEC PF can be bound with vfio, but vfio driver does not support SR-IOV configuration right out of the box, so it will need to be patched.

3.4.2 Enable Virtual Functions

Now, it should be visible in the printouts that PCI PF is under igb_uio control "`Kernel driver in use: igb_uio`"

To show the number of available VFs on the device, read `sriov_totalvfs` file..

```
cat /sys/bus/pci/devices/0000\:<b>\:<d>.<f>/sriov_totalvfs

where 0000\:<b>\:<d>.<f> is the PCI device ID
```

To enable VFs via igb_uio, echo the number of virtual functions intended to enable to `max_vfs` file..

```
echo <num-of-vfs> > /sys/bus/pci/devices/0000\:<b>\:<d>.<f>/max_vfs
```

Afterwards, all VFs must be bound to appropriate UIO drivers as required, same way it was done with the physical function previously.

Enabling SR-IOV via vfio driver is pretty much the same, except that the file name is different:

```
echo <num-of-vfs> > /sys/bus/pci/devices/0000\:<b>\:<d>.<f>/sriov_numvfs
```

3.4.3 Configure the VFs through PF

The PCI virtual functions must be configured before working or getting assigned to VMs/Containers. The configuration involves allocating the number of hardware queues, priorities, load balance, bandwidth and other settings necessary for the device to perform FEC functions.

This configuration needs to be executed at least once after reboot or PCI FLR and can be achieved by using the function `fpga_lte_fec_configure()`, which sets up the parameters defined in `fpga_lte_fec_conf` structure:

```
struct fpga_lte_fec_conf {
    bool pf_mode_en;
    uint8_t vf_ul_queues_number[FPGA_LTE_FEC_NUM_VFS];
    uint8_t vf_dl_queues_number[FPGA_LTE_FEC_NUM_VFS];
    uint8_t ul_bandwidth;
    uint8_t dl_bandwidth;
    uint8_t ul_load_balance;
    uint8_t dl_load_balance;
    uint16_t flr_time_out;
};
```

- `pf_mode_en`: identifies whether only PF is to be used, or the VFs. PF and VFs are mutually exclusive and cannot run simultaneously. Set to 1 for PF mode enabled. If PF mode is enabled all queues available in the device are assigned exclusively to PF and 0 queues given to VFs.
- `vf_*l_queues_number`: defines the hardware queue mapping for every VF.
- `*l_bandwidth`: in case of congestion on PCIe interface. The device allocates different bandwidth to UL and DL. The weight is configured by this setting. The unit of weight is 3 code blocks. For example, if the code block cbps (code block per second) ratio between UL and DL is 12:1, then the configuration value should be set to 36:3. The schedule algorithm is based on code block regardless the length of each block.
- `*l_load_balance`: hardware queues are load-balanced in a round-robin fashion. Queues get filled first-in first-out until they reach a pre-defined watermark level, if exceeded, they won't get assigned new code blocks.. This watermark is defined by this setting.

  If all hardware queues exceeds the watermark, no code blocks will be streamed in from UL/DL code block FIFO.
- `flr_time_out`: specifies how many 16.384us to be FLR time out. The time_out = flr_time_out x 16.384us. For instance, if you want to set 10ms for the FLR time out then set this setting to 0x262=610.

An example configuration code calling the function `fpga_lte_fec_configure()` is shown below:

```
struct fpga_lte_fec_conf conf;
unsigned int i;

memset(&conf, 0, sizeof(struct fpga_lte_fec_conf));
conf.pf_mode_en = 1;

for (i = 0; i < FPGA_LTE_FEC_NUM_VFS; ++i) {
    conf.vf_ul_queues_number[i] = 4;
    conf.vf_dl_queues_number[i] = 4;
}
conf.ul_bandwidth = 12;
conf.dl_bandwidth = 5;
conf.dl_load_balance = 64;
conf.ul_load_balance = 64;

/* setup FPGA PF */
ret = fpga_lte_fec_configure(info->dev_name, &conf);
TEST_ASSERT_SUCCESS(ret,
    "Failed to configure 4G FPGA PF for bbdev %s",
    info->dev_name);
```

3.5 Test Application

BBDEV provides a test application, `test-bbdev.py` and range of test data for testing the functionality of FPGA LTE FEC turbo encode and turbo decode, depending on the device's capabilities. The test application is located under app->test-bbdev folder and has the following options:

```
"-p", "--testapp-path": specifies path to the bbdev test app.
"-e", "--eal-params"  : EAL arguments which are passed to the test app.
"-t", "--timeout"     : Timeout in seconds (default=300).
"-c", "--test-cases"  : Defines test cases to run. Run all if not specified.
"-v", "--test-vector" : Test vector path (default=dpdk_path+/app/test-bbdev/test_vectors/bbdev_
"-n", "--num-ops"     : Number of operations to process on device (default=32).
"-b", "--burst-size"  : Operations enqueue/dequeue burst size (default=32).
```

```
"-l", "--num-lcores"  : Number of lcores to run (default=16).
"-i", "--init-device" : Initialise PF device with default values.
```

To execute the test application tool using simple turbo decode or turbo encode data, type one of the following:

```
./test-bbdev.py -c validation -n 64 -b 8 -v ./turbo_dec_default.data
./test-bbdev.py -c validation -n 64 -b 8 -v ./turbo_enc_default.data
```

The test application `test-bbdev.py`, supports the ability to configure the PF device with a default set of values, if the "-i" or "- -init-device" option is included. The default values are defined in test_bbdev_perf.c as:

- VF_UL_QUEUE_VALUE 4
- VF_DL_QUEUE_VALUE 4
- UL_BANDWIDTH 3
- DL_BANDWIDTH 3
- UL_LOAD_BALANCE 128
- DL_LOAD_BALANCE 128
- FLR_TIMEOUT 610

3.5.1 Test Vectors

In addition to the simple turbo decoder and turbo encoder tests, bbdev also provides a range of additional tests under the test_vectors folder, which may be useful. The results of these tests will depend on the FPGA LTE FEC capabilities:

- **turbo decoder tests:**
  - `turbo_dec_c1_k6144_r0_e10376_crc24b_sbd_negllr_high_snr.data`
  - `turbo_dec_c1_k6144_r0_e10376_crc24b_sbd_negllr_low_snr.data`
  - `turbo_dec_c1_k6144_r0_e34560_negllr.data`
  - `turbo_dec_c1_k6144_r0_e34560_sbd_negllr.data`
  - `turbo_dec_c2_k3136_r0_e4920_sbd_negllr_crc24b.data`
  - `turbo_dec_c2_k3136_r0_e4920_sbd_negllr.data`
- **turbo encoder tests:**
  - `turbo_enc_c1_k40_r0_e1190_rm.data`
  - `turbo_enc_c1_k40_r0_e1194_rm.data`
  - `turbo_enc_c1_k40_r0_e1196_rm.data`
  - `turbo_enc_c1_k40_r0_e272_rm.data`
  - `turbo_enc_c1_k6144_r0_e18444.data`
  - `turbo_enc_c1_k6144_r0_e32256_crc24b_rm.data`
  - `turbo_enc_c2_k5952_r0_e17868_crc24b.data`
  - `turbo_enc_c3_k4800_r2_e14412_crc24b.data`

```
    - turbo_enc_c4_k4800_r2_e14412_crc24b.data
```