

DPDK&PMD development on ARMv8

- base on Ring buffer DMA on zcu102

Alex He(ahe@xilinx.com)

TEMA Embedded Application Engineer

Nov/1/2018



DPDK Introduction



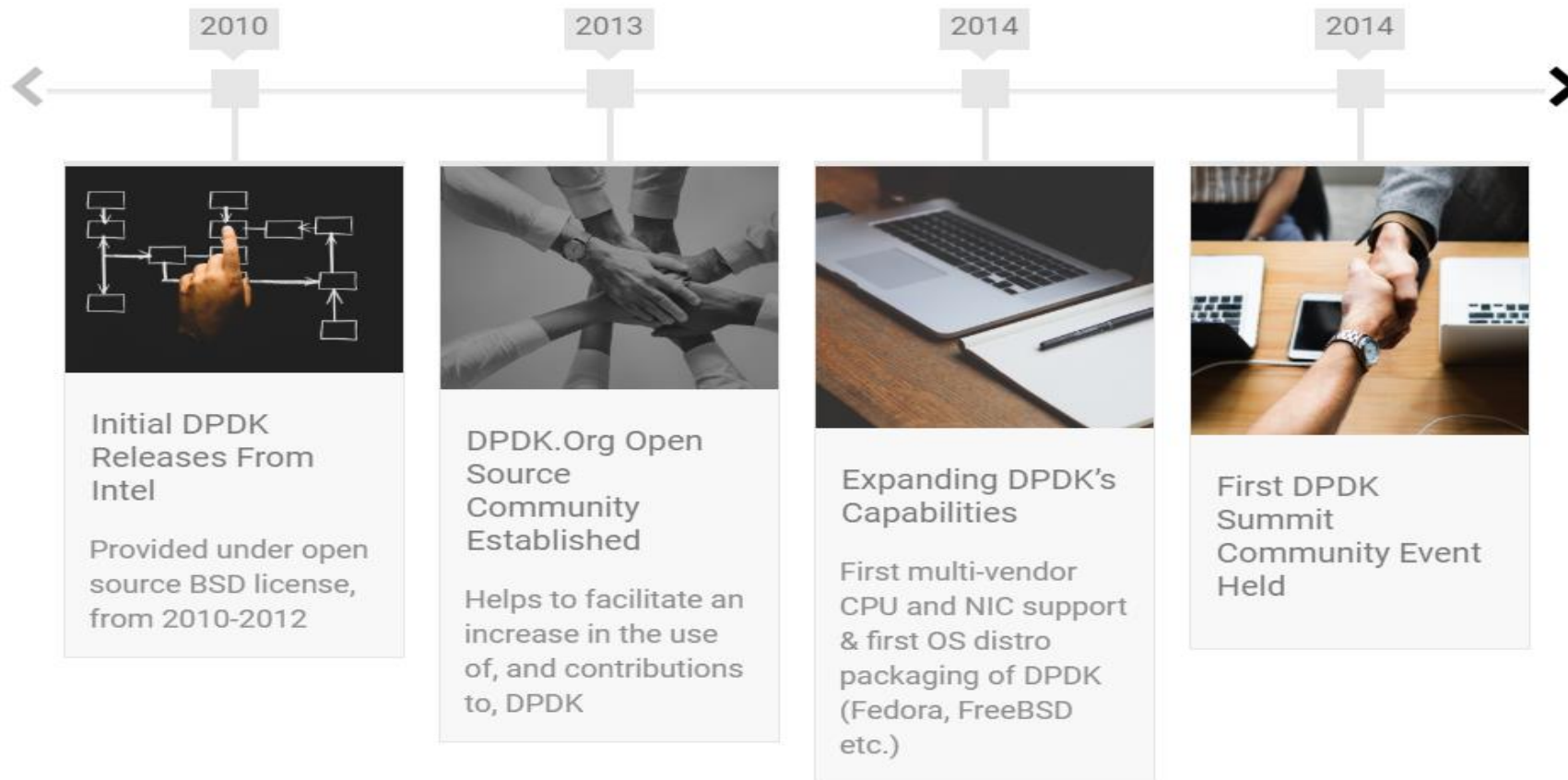
ABOUT DPDK

Main Goal

is to provide a simple, complete framework for **fast packet processing** in data plane applications

- > Homepage: <https://www.dpdk.org/>
- > DPDK is the **Data Plane Development Kit**
- > Consists of **libraries** to accelerate packet processing workloads
- > Running on a wide variety of CPU architectures. (**x86**, **POWER** and **ARM**)
- > Run mostly in **Linux** user land, with a **FreeBSD** port available for a subset of DPDK features
- > DPDK is licensed under the **Open Source BSD** License

History of DPDK



DPDK now supports all major CPU architectures and NICs from multiple vendors, which makes it ideally suited to applications that need to be portable across multiple platforms.

DPDK Release (until 2018 July)

FILE	DATE/CHANGELOG
DPDK 18.05	2018 May 30
DPDK 18.02.2	2018 June 15
DPDK 17.11.3 (LTS)	2018 June 15
DPDK 17.08.2	2018 April 23
DPDK 17.05.2	2017 September 7
DPDK 17.02.1	2017 June 2
DPDK 16.11.7 (LTS)	2018 June 14
DPDK 16.07.2	2016 November 30
DPDK 16.04	2016 April 12
DPDK 2.2	2015 December 15
DPDK 2.1	2015 August 17
DPDK 2.0	2015 April 3
DPDK 1.8	2014 December 3

Contributor's Guidelines

The **Stable Releases** is to maintain releases of DPDK with backported fixes over an extended period of time.

The **Long Term Support release (LTS)** is a designation applied to a Stable Release to indicate longer term support.

Roadmap (v18.08, 2018 August)

<http://core.dpdk.org/roadmap/>

Downloads

<http://static.dpdk.org/rel/>

<http://git.dpdk.org/dpdk-stable/refs>

Contributing Code to DPDK

The DPDK **development process** is modelled (loosely) on the **Linux Kernel development model**

- The code is hosted in a public git repository.
- There is a mailing list where developers submit patches.
- There are maintainers for hierarchical components.
- Patches are reviewed publicly on the mailing list.
- Successfully reviewed patches are merged to the repository.
- Patches should be sent to the target repository or sub-tree, see below.
- All sub-repositories are merged into main repository for **-rc1** and **-rc2** versions of the release.
- After the **-rc2** release all patches should target the main repository.

Refer to

[Contributor's Guidelines](#)

Source License

- > The DPDK uses the Open Source BSD-3-Clause license for the core libraries and drivers.
- > The kernel components are GPL-2.0 licensed.
- > DPDK uses **single line** reference to **Unique License Identifiers** in source files as defined by the Linux Foundation's [SPDX project](#).

Software Package Data Exchange® (SPDX®)

is an open standard for communicating software bill of material information (including components, licenses, copyrights, and security references).



SPDX License List

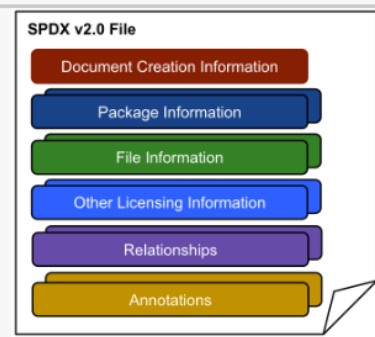
Version: 2.0.0, July 2019
Copyright © 2019 Linux Foundation. All rights reserved.

SPDX License	SPDX License	SPDX License	SPDX License
Apache License 2.0	Apache License 2.0	Apache License 2.0	Apache License 2.0
GNU General Public License v2.0	GNU General Public License v2.0	GNU General Public License v2.0	GNU General Public License v2.0
GNU General Public License v3.0	GNU General Public License v3.0	GNU General Public License v3.0	GNU General Public License v3.0
GNU Lesser General Public License v2.1	GNU Lesser General Public License v2.1	GNU Lesser General Public License v2.1	GNU Lesser General Public License v2.1
GNU Lesser General Public License v3.0	GNU Lesser General Public License v3.0	GNU Lesser General Public License v3.0	GNU Lesser General Public License v3.0
MIT License	MIT License	MIT License	MIT License
BSD 3-Clause License	BSD 3-Clause License	BSD 3-Clause License	BSD 3-Clause License
BSD 2-Clause License	BSD 2-Clause License	BSD 2-Clause License	BSD 2-Clause License
CC BY-SA 4.0 International	CC BY-SA 4.0 International	CC BY-SA 4.0 International	CC BY-SA 4.0 International
CC BY-NC-SA 4.0 International	CC BY-NC-SA 4.0 International	CC BY-NC-SA 4.0 International	CC BY-NC-SA 4.0 International
CC BY-ND 4.0 International	CC BY-ND 4.0 International	CC BY-ND 4.0 International	CC BY-ND 4.0 International
CC BY-NC-ND 4.0 International	CC BY-NC-ND 4.0 International	CC BY-NC-ND 4.0 International	CC BY-NC-ND 4.0 International
CC0 1.0 Universal	CC0 1.0 Universal	CC0 1.0 Universal	CC0 1.0 Universal
Public Domain	Public Domain	Public Domain	Public Domain
Unlicense	Unlicense	Unlicense	Unlicense

SPDX License Identifiers in Source

```
/*  
SPDX-License-Identifier: BSD-3-Clause  
Copyright ...
```

SPDX Documents



Source License (Con.)

DPDK uses first line of the file to be SPDX tag. In case of `#!/` scripts, SPDX tag can be placed in 2nd line of the file.

Example:

1. To label a file as subject to the BSD-3-Clause license

SPDX-License-Identifier: BSD-3-Clause

2. To label a file as dual-licensed with BSD-3-Clause and GPL-2.0
(e.g., for code that is shared between the kernel and userspace)

SPDX-License-Identifier: (BSD-3-Clause OR GPL-2.0)

Xilinx PMD follow it and add Copyright of Xilinx

```
/* SPDX-License-Identifier: BSD-3-Clause  
 * Copyright(c) 2018 Xilinx, Inc  
 */
```


Key Concepts & Core Components

■ Environment Abstraction Layer(EAL)

provides a generic interface that hides the environment specifics from the applications and libraries.

- DPDK loading and launching
- Support for multi-process and multi-thread execution types
- Core affinity/assignment procedures
- System memory allocation/de-allocation (using mmap() in hugetlbfs, Mempool Library)
- Memory management (malloc-like functions with hugepage memory)
- Atomic/lock operations
- Time reference
- PCI bus access (Through /sys of linux kernel)
- Trace and debug functions (log sub-system, debug configs)
- CPU feature identification
- Interrupt handling (register/unregister callbacks, depend on UIO)
- Alarm operations

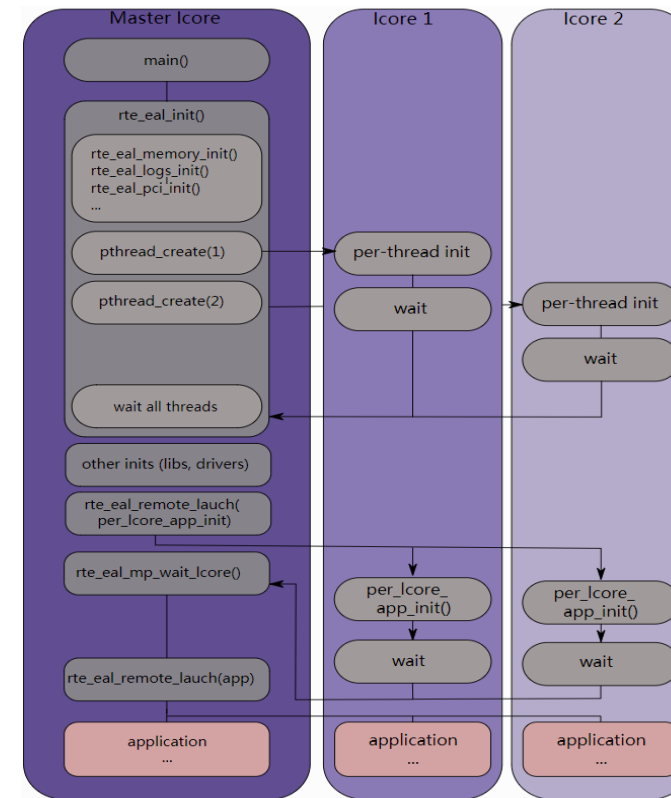


Fig. 3.1 EAL Initialization in a Linux Application Environment

Key Concepts & Core Components (Con.)

■ Poll Mode Driver(PMD)

- consists of APIs defined by the DPDK PMD framework
- running in **user space**
- to configure the devices and their respective queues.
- In addition, a PMD accesses the RX and TX descriptors directly without any interrupts (with the exception of Link Status Change interrupts) to quickly receive, process and deliver packets in the user's application.

Do what the kernel driver does!

Looks like kernel driver!

```
static struct rte_vdev_driver pmd_xlnx_drv = {  
    .probe = rte_pmd_xlnx_probe,  
    .remove = rte_pmd_xlnx_remove,  
};
```

```
struct rte_vdev_driver {  
    TAILQ_ENTRY(rte_vdev_driver) next; /**< Next in list. */  
    struct rte_driver driver; /**< Inherited general driver. */  
    rte_vdev_probe_t *probe; /**< Virtual device probe function. */  
    rte_vdev_remove_t *remove; /**< Virtual device remove function. */  
};
```

Key Concepts & Core Components (Con.)

■ mbuf(struct rte_mbuf)

- ❑ Represent the packet (like skb in linux kernel)
- ❑ Metadata + payload data (from/to HW)

■ mbuf library(APIs)

- ❑ Allocate and free buffers (mbufs)
- ❑ Manipulating mbufs
 - Get data length
 - Get a pointer to the start of data
 - Prepend data before data
 - Append data after data
 - Remove data at the beginning of the buffer (`rte_pktmbuf_adj()`)
 - Remove data at the end of the buffer (`rte_pktmbuf_trim()`) Refer to the *DPDK API Reference* for details.

■ **memory pool**

- ❑ allocator of a fixed-sized object
- ❑ mbuf depend on it

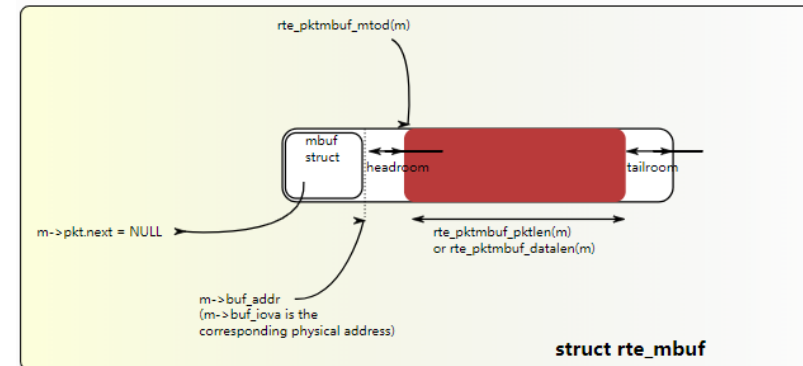


Fig. 7.1 An mbuf with One Segment

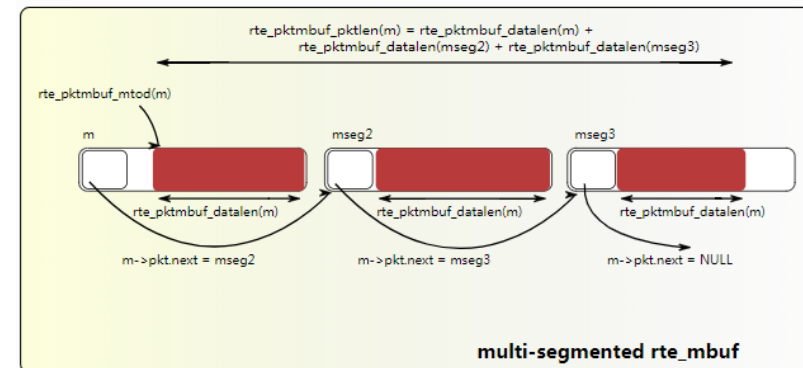
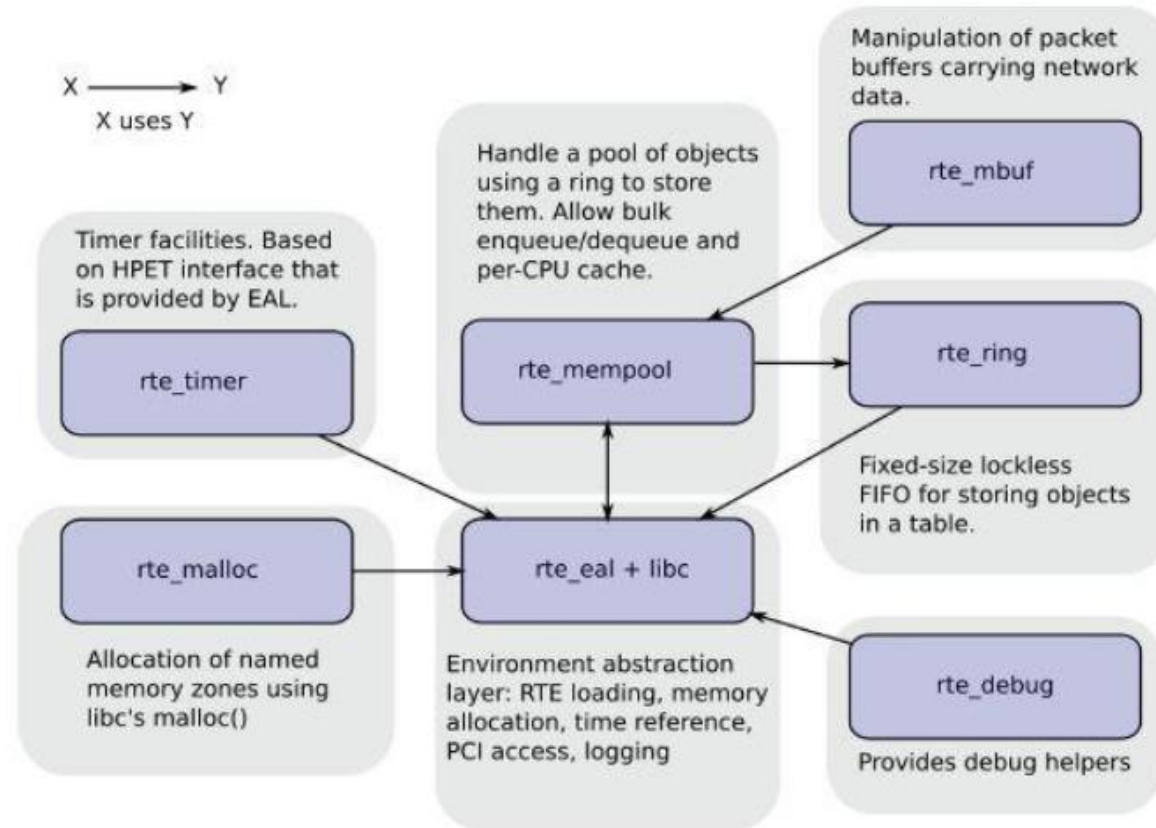


Fig. 7.2 An mbuf with Three Segments

Key Concepts & Core Components

Relationship of DPDK Libraries



<https://feisky.gitbooks.io/sdn/dpdk/>

PMD development

XPMD for Ring buffer DMA(RDMA) IP



Example Platform

- > **Xilinx PMD for Ring buffer DMA(RDMA) IP**
 - >> XPMD/RDMA (abbreviations)
- > **ZCU102 (CortexA53, Armv8)**
- > **BSP: Petalinux v2017.4**
- > **gcc version 6.2.1 20161114 (Linaro GCC Snapshot 6.2-2016.11)**
- > **DPDK 18.02**

Key points of XPMD implement

Challenges:

- > Armv8(short of documents)
- > Without UIO(directly access/mmap registers, Phy2Virt address)
- > Virtual address to Physical address for DMA operation
- > Cache coherent (no HPC port connect to DMA)
- > Debug/Test
- > DMA IP is still under developing meanwhile to be mature
- > Performance
- > ...

We do it almost from ZERO!

Benefit from Linux Kernel Driver Experience!

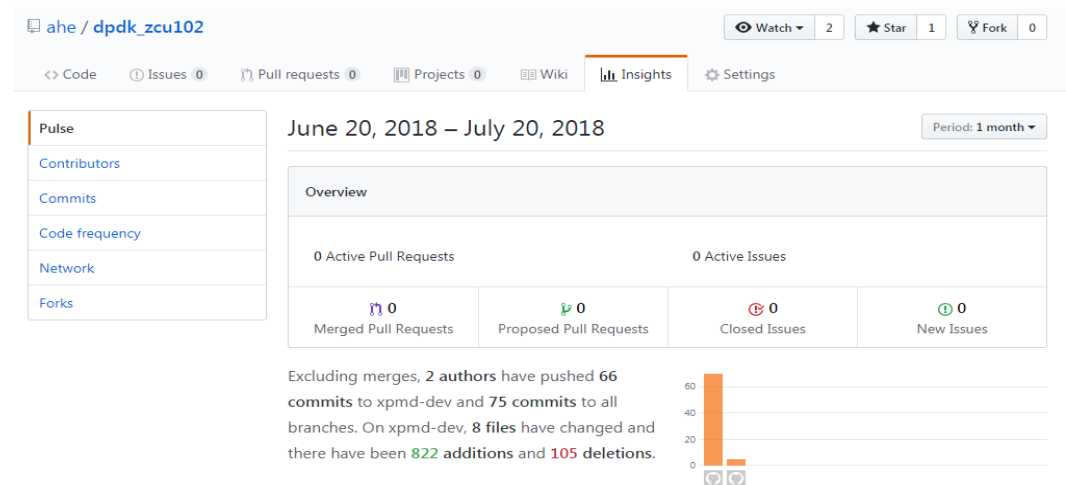
Development process

- > Use git for code management

https://github.com/alexhegit/dpdk_zcu102

Open source development flow(branch, patches)

- > No upstream until now



XPMD

- ❑ Main work is implement the eth_dev_ops and tx/rx burst functions
- ❑ More like the Linux kernel driver
- ❑ Object-Oriented data structure
- ❑ Leverage EAL APIs
- ❑ Driver and device binding
- ❑ Driver support input parameters
 - get the register physical address

```
ahe@ahe-5810:~/DPDK_ARM64/dpdk$ ls -al drivers/net/xlnx/
total 68
drwxrwxr-x  2 ahe ahe  4096 7月  20 15:25 .
drwxrwxr-x 40 ahe ahe  4096 6月  7 11:26 ..
-rw-rw-r--  1 ahe ahe  2045 6月  7 11:26 Makefile
-rw-rw-r--  1 ahe ahe   113 6月  7 11:26 meson.build
-rw-rw-r--  1 ahe ahe 25411 7月 19 16:55 rte_eth_xlnx.c
-rw-rw-r--  1 ahe ahe    28 6月  7 11:26 rte_pmd_xlnx_version.map
-rw-rw-r--  1 ahe ahe   761 6月 20 15:30 xlnx_logs.h
-rw-rw-r--  1 ahe ahe  2801 7月 19 16:55 xlnx_rdma.h
-rw-rw-r--  1 ahe ahe  4351 7月 12 11:10 xlnx_rdma_reg.h
-rw-rw-r--  1 ahe ahe  1619 7月 20 14:43 xpmd_readme
ahe@ahe-5810:~/DPDK_ARM64/dpdk$
```

```
static const struct eth_dev_ops ops = {
    .dev_start = eth_dev_start,
    .dev_stop = eth_dev_stop,
    .dev_close = eth_dev_close,
    .dev_configure = eth_dev_configure,
    .dev_infos_get = eth_dev_info,
    .rx_queue_setup = eth_rx_queue_setup,
    .tx_queue_setup = eth_tx_queue_setup,
    .rx_queue_release = eth_rx_queue_release,
    .tx_queue_release = eth_tx_queue_release,
    .mtu_set = eth_mtu_set,
    .link_update = eth_link_update,
    .mac_addr_set = eth_mac_address_set,
    .stats_get = eth_stats_get,
    .stats_reset = eth_stats_reset,
};
```

```
/* finally assign rx and tx ops */
eth_dev->rx_pkt_burst = eth_xlnx_rx;
eth_dev->tx_pkt_burst = eth_xlnx_tx;
```

Key points

Armv8(short of documents)

- > Create new config(./config/defconfig_arm64-armv8a-plnxapp-gcc) base on common_armv8_linuxapp
- > disable NUMA

```
--- a/config/defconfig_arm64-armv8a-plnxapp-gcc
+++ b/config/defconfig_arm64-armv8a-plnxapp-gcc
@@ -33,3 +33,11 @@
 
 CONFIG_RTE_TOOLCHAIN="gcc"
 CONFIG_RTE_TOOLCHAIN_GCC=y
+
+#
+# Alex's Config
+#
+# Doesn't support NUMA
+CONFIG_RTE_EAL_NUMA_AWARE_HUGEPAGES=n
+CONFIG_RTE_LIBRTE_VHOST_NUMA=n
```

Key points(Con.)

Without UIO(register Physical address to Virtual address)

> Input parameter of the base register Physical address

`$testpmd -l 1-3 --vdev 'net_xlnx0, pbase=0xa0000000 -- -i`

> mmap it through /dev/mem

```
static void
xlnx_regs_mmap(struct rdma_dev *rdma_dev)
{
    int fd;

    fd = open("/dev/mem", O_RDWR | O_SYNC);

    rdma_dev->regs_vbase = mmap(NULL, 4096,
                                PROT_READ | PROT_WRITE, MAP_SHARED,
                                fd, rdma_dev->regs_pbase);

    close(fd);
}
```

Key points(Con.)

Register access

- > DPDK provide APIs for 8/16/32/64 bit register access

Low lever with ASM code in *rte_read*_relaxed()/rte_write*_relaxed()*

High lever with memory barrier in *rte_read*()/rte_write*()*

- > Wrapper these APIs for easy coding in PMD

- > Or call the high lever APIs directly in PMD

```
static __rte_always_inline uint32_t
rte_read32_relaxed(const volatile void *addr)
{
    uint32_t val;

    asm volatile(
        "ldr %w[val], [%x[addr]]"
        : [val] "=r" (val)
        : [addr] "r" (addr));

    return val;
}
```

```
static __rte_always_inline uint32_t
rte_read32(const volatile void *addr)
{
    uint32_t val;
    val = rte_read32_relaxed(addr);
    rte_io_rmb();
    return val;
}
```

```
static __rte_always_inline void
rte_write32_relaxed(uint32_t val, volatile void *addr)
{
    asm volatile(
        "str %w[val], [%x[addr]]"
        :
        : [val] "r" (val), [addr] "r" (addr));
}
```

```
static __rte_always_inline void
rte_write32(uint32_t value, volatile void *addr)
{
    rte_io_wmb();
    rte_write32_relaxed(value, addr);
}
```

Key points(Con.)

Virtual address to Physical address

- > Use RTE API to allocate the memory with virtual address back
 - >> The last parameters of `rte_zmalloc()` will keep the memory align as what your DMA want
- > Use RTE API do the `virt2phy` address

```
rxq->ring_vaddr = rte_zmalloc("rxq->ring_vaddr",
                              sizeof(union rdma_rx_desc) * nb_rx_desc,
                              XLNX_RDMA_MEM_ALIGN);
if (!rxq->ring_vaddr) {
    RTE_LOG(ERR, PMD, "failed to alloc mem for rx ring\n");
    return -ENOMEM;
}
rxq->ring_paddr = rte_mem_virt2phy(rxq->ring_vaddr);
```

- > Use RTE MBUF APIs to get packet data physical directly for DMA descriptor

```
/*
 * Put bufs to DMA TX ring
 * and update mbufs_info
 */
hw_p = txq->hw_p;
for (i = 0; i < send_mbuf_num; i++) {
    tx_desc = (union rdma_tx_desc *)txq->ring_vaddr + hw_p;
    txq->mbufs_info[hw_p] = bufs[i];
    tx_desc->read.pkt_addr = rte_mbuf_data_iova(bufs[i]);
    tx_desc->read.pkt_size = rte_pktmbuf_data_len(bufs[i]);
    tx_desc->read.seop.sop = 0x1;
    tx_desc->read.seop.eop = 0x1;
    tx_desc->read.rsvd3 = 0x1;
    hw_p = (hw_p + 1) & (txq->ring_size - 1);
}
```

Key points(Con.)

Cache coherent (no HPC port connect to DMA)

> Call `__clear_cache()`

```
89  
90 static void  
91 rdma_flush_ring(struct rdma_queue *q)  
92 {  
93     uint64_t start, stop;  
94  
95     start = (uint64_t)q->ring_vaddr;  
96     stop = (uint64_t)q->ring_vend;  
97  
98     //invalidate dcache_range(start, stop);  
99     clear_cache((char *)start, (char *)stop);  
100 }  
101 }  
102
```

Refer to: <http://gcc.gnu.org/onlinedocs/gcc/Other-Builtins.html>

Key points(Con.)

Debug/Test

- > Register special log functions
- > Enable the log by the parameters
- > Use testpmd for test
 - >> Original strong test application of DPDK
 - >> Good example of user application for packet forward
 - >> [Testpmd Application User Guide](http://doc.dpdk.org/guides/testpmd_app_ug/index.html)

```
e.g.
$testpmd -l 1-3 -n 4 --log-level=pmd.net.xlnx.init,8 --log-level=pmd.net.xlnx.driver,8 --vdev 'net_xlnx0, pbase=0xa0000000 -- -i

Args:
  --log-level: set print out information level.
  pbase: start register's physical address

Please refer to http://doc.dpdk.org/guides/testpmd\_app\_ug/index.html for more details of testpmd
```

```
commit 5fee7d8d1e69cffeaddcb27e6add98cef062df
Author: Alex He <ahe@xilinx.com>
Date: Wed Jun 6 15:29:20 2018 +0800

    xpm: Add log functions

    Signed-off-by: Alex He <ahe@xilinx.com>

diff --git a/drivers/net/xlnx/rte_eth_xlnx.c b/drivers/net/xlnx/rte_eth_xlnx.c
index 1a5c248..e738d8e 100644
--- a/drivers/net/xlnx/rte_eth_xlnx.c
+++ b/drivers/net/xlnx/rte_eth_xlnx.c
@@ -1,5 +1,5 @@
/*
 * SPDX-License-Identifier: BSD-3-Clause
 * Copyright(c) 2018 Xilinx
 * Copyright(c) 2018 Xilinx, Inc
 */

#include <rte_mbuf.h>
@@ -58,6 +58,23 @@ static struct rte_eth_link pmd_link = {
    .link_autoneg = ETH_LINK_AUTONEG,
};

+int xlnx_net_logtype_init;
+int xlnx_net_logtype_driver;
+RTE_INIT(xlnx_net_init_log);
+static void
+xlnx_net_init_log(void)
+{
+    xlnx_net_logtype_init = rte_log_register("pmd.net.xlnx.init");
+    if (xlnx_net_logtype_init >= 0)
+        rte_log_set_level(xlnx_net_logtype_init, RTE_LOG_NOTICE);
+
+    xlnx_net_logtype_driver = rte_log_register("pmd.net.xlnx.driver");
+    if (xlnx_net_logtype_driver >= 0)
+        rte_log_set_level(xlnx_net_logtype_driver, RTE_LOG_NOTICE);
+}
+
static uint16_t
eth_xlnx_rx(void *q, struct rte_mbuf **bufs, uint16_t nb_bufs)
{
diff --git a/drivers/net/xlnx/xlnx_logs.h b/drivers/net/xlnx/xlnx_logs.h
new file mode 100644
index 0000000..dc0dfce
--- /dev/null
+++ b/drivers/net/xlnx/xlnx_logs.h
@@ -0,0 +1,24 @@
/*
 * SPDX-License-Identifier: BSD-3-Clause
 * Copyright(c) 2018 Xilinx, Inc
 */

#ifndef __XLNX_LOGS_H__
#define __XLNX_LOGS_H__

#define PMD_INIT_LOG(level, fmt, args...) \
    rte_log(RTE_LOG_## level, xlnx_net_logtype_init, \
            "%s(): " fmt "\n", __func__, ## args)

#define PMD_INIT_FUNC_TRACE() PMD_INIT_LOG(DEBUG, ">>")

#define PMD_DRV_LOG(level, fmt, args...) \
    rte_log(RTE_LOG_## level, xlnx_net_logtype_driver, \
            "%s(): " fmt "\n", __func__, ## args)

#define xlnx_log_err(s, ...) PMD_INIT_LOG(ERR, s, ## __VA_ARGS__)
#define xlnx_log_dbg(s, ...) PMD_DRV_LOG(DEBUG, s, ## __VA_ARGS__)

extern int xlnx_net_logtype_init;
extern int xlnx_net_logtype_driver;

#endif /* __XLNX_LOGS_H__ */
```

Key points(Con.)

Optimization

- > Use & operator replace % for save cpu instruction cycle of ring round
- > Keep struct rdma_dev and rdma_queue cache aligned
- > Count the pkt.cn directly since no parallel access(multi-thread)

May Still have space to do MORE both HW and SW

```
commit 7bf49de132299fa61ef9e500ed81c753adc2700b
Author: Alex He <ahe@xilinx.com>
Date: Thu Jul 19 15:10:09 2018 +0800

    xpmc:opt: use & operator do ring pointer round

    Aligns input parameter of ring size to next power of 2. Then we replace
    the % operator with & to do the ring pointer round move for less cpu
    instructions and better performance.

    Signed-off-by: Alex He <ahe@xilinx.com>

commit c5147eeled8893a66ab9fc68c79eda520a86eb15
Author: Alex He <ahe@xilinx.com>
Date: Thu Jul 19 10:49:03 2018 +0800

    xpmc:opt: keep struct rdma_dev cache aligned

    To get good performance from it.

    Signed-off-by: Alex He <ahe@xilinx.com>

commit f32ddcec40badaa3af09b9c6fd9775dd1d651a39
Author: Alex He <ahe@xilinx.com>
Date: Thu Jul 19 10:37:00 2018 +0800

    xpmc:opt: keep the struct rdma_queue cache aligned

    The rdma_queue is accessed very frequent. The performance may benfit
    from keeping it cache aligned.

    Signed-off-by: Alex He <ahe@xilinx.com>

commit c801d28c5aed14da4baf08bad057b5ba0cde58a4
Author: Alex He <ahe@xilinx.com>
Date: Thu Jul 19 10:00:26 2018 +0800

    xpmc:opt: count pkts.cn directly

    Signed-off-by: Alex He <ahe@xilinx.com>
```


How to Test

testpmd:

```
$testpmd -l 1-3 --vdev 'net_xlnx0, pbase=0xa0000000 -- -i
```

"-i" means interactive mode

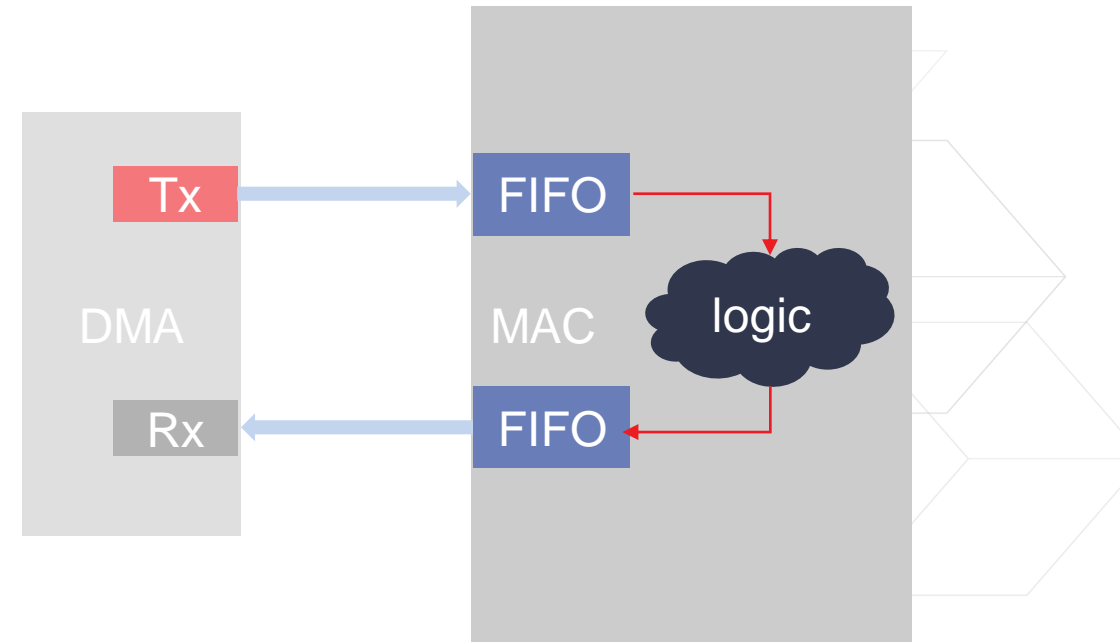
```
$start tx_first
```

...transfer start...

```
$show port stats all
```

...get the statistic report ...

```
$stop
```



Loop back in HW

Performance Test Report

- > **One port(DMA) can reach to 0.8M PPS** feedback from customer
- > **Test reports from our side(with HW IP updated from Jason Wu)**

NO.	max-pkt-len	burst	PPS	CPU loading /zcu102	Memory/hugepages
1	1518	32	697K	25% of 4cores,100% - 1core	1077MB
2	1518	64	966K	25% of 4cores,100% - 1core	1077MB
3	1518	128	1166K	25% of 4cores,100% - 1core	1077MB
4	1518	256	1285K	25% of 4cores,100% - 1core	1077MB
5	1518	512	1347K	25% of 4cores,100% - 1core	1077MB
6	64	32	697K	25% of 4cores,100% - 1core	1077MB
7	64	64	966K	25% of 4cores,100% - 1core	1077MB
8	64	128	1167K	25% of 4cores,100% - 1core	1077MB
9	64	256	1285K	25% of 4cores,100% - 1core	1077MB
10	64	512	1347K	25% of 4cores,100% - 1core	1077MB

Reference & Resource

- > <https://www.dpdk.org/>
- > <http://core.dpdk.org/doc/>
- > <https://spdx.org/>
- > 《深入浅出DPDK》
 - >> Main authors are Intel DPDK experts in China

Git

- > https://github.com/alexhegit/dpdk_zcu102



Adaptable.
Intelligent.

