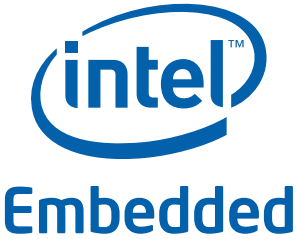

Intel® Data Plane Development Kit - Timer Sample Application

User Guide

April 2012

Intel Confidential

Reference Number: 482254-1.1

Contents

Revision History

Date	Revision	Description
March 2012	1.1	Updates for software release 1.2
September 2011	1.0	Initial release

1.0 Description

The Timer sample application is a simple application that demonstrates the use of a timer in an Intel® DPDK application. This application prints some messages from different lcores regularly, demonstrating the use of timers.

1.1 Documentation Roadmap

The following is a list of Intel® DPDK documents in suggested reading order:

- **Release Notes**: Provides release-specific information, including supported features, limitations, fixed issues, known issues and so on. Also, provides the answers to frequently asked questions in FAQ format.
- **Getting Started Guide**: Describes how to install and configure the Intel® DPDK software; designed to get users up and running quickly with the software.
- **Programmer's Guide**: Describes:
  - The software architecture and how to use it (through examples), specifically in a Linux* application (linuxapp) environment
  - The content of the Intel® DPDK, the build system (including the commands that can be used in the root Intel® DPDK Makefile to build the development kit and an application) and guidelines for porting an application
  - Optimizations used in the software and those that should be considered for new development

  A glossary of terms is also provided.
- **API Reference**: Provides detailed information about Intel® DPDK functions, data structures and other programming constructs.
- **Sample Application User Guides**: A set of guides, each describing a sample application that showcases specific functionality, together with instructions on how to compile, run and use the sample application.

2.0 Compiling the Application

1. Go to the example directory:

   ```
   export RTE_SDK=/path/to/rte_sdk
   cd ${RTE_SDK}/examples/timer
   ```

2. Set the target (a default target is used if not specified). For example:

   ```
   export RTE_TARGET=x86_64-default-linuxapp-gcc
   ```

   See the *Intel® DPDK Getting Started Guide* for possible `RTE_TARGET` values.

3. Build the application:

   ```
   make
   ```

3.0 Running the Application

To run the example in linuxapp environment:

```
$ ./build/timer -c f -n 4
```

Refer to the *Intel® DPDK Getting Started Guide* for general information on running applications and the Environment Abstraction Layer (EAL) options.

4.0 Explanation

The following sections provide some explanation of the code.

4.1 Initialization and Main Loop

In addition to EAL initialization, the timer subsystem must be initialized, by calling the `rte_timer_subsystem_init()` function.

```
/* init EAL */
ret = rte_eal_init(argc, argv);
if (ret < 0)
    rte_panic("Cannot init EAL\n");

/* init RTE timer library */
rte_timer_subsystem_init();
```

After timer creation (see the next paragraph), the main loop is executed on each slave lcore using the well-known `rte_eal_remote_launch()` and also on the master.

```
/* call lcore_mainloop() on every slave lcore */
RTE_LCORE_FOREACH_SLAVE(lcore_id) {
rte_eal_remote_launch(lcore_mainloop, NULL, lcore_id);
}

/* call it on master lcore too */
lcore_mainloop(NULL);
```

The main loop is very simple in this example:

```
while (1) {
    /*
     * Call the timer handler on each core: as we don't
     * need a very precise timer, so only call
     * rte_timer_manage() every ~10ms (at 2 Ghz). In a real
     * application, this will enhance performances as
     * reading the HPET timer is not efficient.
     */
    cur_tsc = rte_rdtsc();
    diff_tsc = cur_tsc - prev_tsc;
    if (diff_tsc > TIMER_RESOLUTION_CYCLES) {
        rte_timer_manage();
        prev_tsc = cur_tsc;
    }
}
```

As explained in the comment, it is better to use the *TSC* register (as it is a per-lcore register) to check if the `rte_timer_manage()` function must be called or not. In this example, the resolution of the timer is 10 milliseconds.

4.2 Managing Timers

In the `main()` function, the two timers are initialized. This call to `rte_timer_init()` is necessary before doing any other operation on the timer structure.

```
/* init timer structures */
rte_timer_init(&timer0);
rte_timer_init(&timer1);
```

Then, the two timers are configured:

- The first timer (`timer0`) is loaded on the master lcore and expires every second. Since the `PERIODICAL` flag is provided, the timer is reloaded automatically by the timer subsystem. The callback function is `timer0_cb()`.
- The second timer (`timer1`) is loaded on the next available lcore every 333 ms. The `SINGLE` flag means that the timer expires only once and must be reloaded manually if required. The callback function is `timer1_cb()`.

```
/* load timer0, every second, on master lcore, reloaded automatically */
hz = rte_get_hpet_hz();
lcore_id = rte_lcore_id();
rte_timer_reset(&timer0, hz, PERIODICAL, lcore_id, timer0_cb, NULL);

/* load timer1, every second/3, on next lcore, reloaded manually */
lcore_id = rte_get_next_lcore(lcore_id, 0, 1);
rte_timer_reset(&timer1, hz/3, SINGLE, lcore_id, timer1_cb, NULL);
```

The callback for the first timer (`timer0`) only displays a message until a global counter reaches 20 (after 20 seconds). In this case, the timer is stopped using the `rte_timer_stop()` function.

```
/* timer0 callback */
static void
timer0_cb(__attribute__((unused)) struct rte_timer *tim,
          __attribute__((unused)) void *arg)
{
    static unsigned counter = 0;
    unsigned lcore_id = rte_lcore_id();

    printf("%s() on lcore %u\n", __FUNCTION__, lcore_id);

    /* this timer is automatically reloaded until we decide to
     * stop it, when counter reaches 20. */
    if ((counter ++) == 20)
        rte_timer_stop(tim);
}
```

The callback for the second timer (`timer1`) displays a message and reloads the timer on the next lcore, using the `rte_timer_reset()` function:

```
/* timer1 callback */
static void
timer1_cb(__attribute__((unused)) struct rte_timer *tim,
          __attribute__((unused)) void *arg)
{
    unsigned lcore_id = rte_lcore_id();
    uint64_t hz;

    printf("%s() on lcore %u\n", __FUNCTION__, lcore_id);
```

```
    /* reload it on another lcore */
    hz = rte_get_hpet_hz();
    lcore_id = rte_get_next_lcore(lcore_id, 0, 1);
    rte_timer_reset(&timer1, hz/3, SINGLE, lcore_id, timer1_cb, NULL);
}
```

§ §