DEPLOYMENT GUIDE

Intel Corporation

Guidelines for Optimizing Power Consumption of vCMTS Deployments on Intel® Xeon® Scalable Processors

Authors

Rory Sexton
Intel Senior Software Engineer

David Coyle
Intel Senior Software Engineer

Dominik Przychodni
Intel Network Software Engineer

1 Introduction

This document describes in detail how both C-states and P-states can be used to unlock significant power savings for virtualized cable modem termination system (vCMTS) deployments. The goal of the document is to provide a set of practical, step-by-step instructions detailing how to leverage C-states and P-states within a vCMTS solution to optimize the power draw of Intel® Xeon® Scalable processors dynamically based on real-time network load.

This document is intended for independent software vendors (ISVs) and multiple system operators (MSOs) within the cable industry, who are deploying or planning to deploy vCMTS solutions based on the Data Plane Development Kit (DPDK) [1] on the latest Intel Xeon Scalable processors.

To keep the document concise and limit it to only the material required to enable ISVs and MSOs in the integration of power management with their vCMTS solution on Intel Xeon Scalable processors, this document assumes a general understanding of the power management features of Intel Xeon processors, C-states, and P-states. Further information on C-states and P-states, including a detailed analysis quantifying the power savings achievable in a vCMTS deployment can be found by studying the reference resources listed in Table 2.

Table of Contents

Figures

Tables

Document Revision History

REVISION	DATE	DESCRIPTION
001	June 2023	Initial revision

1.1 Terminology

Table 1. Terminology

ABBREVIATION	DESCRIPTION
API	Application Programming Interface
CPU	Central Processing Unit
DOCSIS	Data Over Cable Service Interface Specification
DPDK	Data Plane Development Kit
HWP	Hardware P-states
Intel® IPM	Intel® Infrastructure Power Manager
ISV	Independent Software Vendor
KPI	Key Performance Indicator
MAC	Media Access Control
MHz	Megahertz
MSO	Multiple System Operator
NIC	Network Interface Card
PMD	Poll Mode Driver
TSC	Timestamp Counter
vCMTS	Virtualized Cable Modem Termination System

1.2 Reference Documentation

Table 2. Reference Documents

REFERENCE		SOURCE
1	Data Plane Development Kit (DPDK)	https://www.dpdk.org
2	CPU Instruction Reference for User Wait Instruction Set	https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
3	DPDK Power Management Library	https://doc.dpdk.org/guides/prog_guide/power_man.html
4	DPDK l3fwd power sample application	https://doc.dpdk.org/guides/sample_app_ug/l3_forward_power_man.html
5	Intel® vCMTS Reference Dataplane	https://www.intel.com/content/www/us/en/developer/topic-technology/open/vcmts-reference-dataplane/overview.html
6	Pause Patch for DPDK Rings	https://patches.dpdk.org/series/27919/mbox
7	Linux CPU Performance Scaling	https://www.kernel.org/doc/html/latest/admin-guide/pm/cpufreq.html
8	intel_pstate CPU Performance Scaling Driver	https://www.kernel.org/doc/html/latest/admin-guide/pm/intel_pstate.html
9	Intel CommsPowerManagement	https://github.com/intel/CommsPowerManagement
10	Intel® Infrastructure Power Manager Solution Brief	https://www.intel.com/content/www/us/en/wireless-network/core-network/infrastructure-power-manager-solution-brief.html

2 Power-Optimized C-states

2.1 Technology Overview

4th Gen Intel Xeon Scalable processors introduce new power-optimized C-states, which are considered substates of C0, and known as C0.1 and C0.2. Like the deeper C-states (C1, C1E, C6), instruction execution stops once a CPU core enters one of these new C-states. However, the new C-states have much lower exit latencies and are, therefore, much better suited to dataplane workloads.

These C-states can be very effective at reducing power consumption for DPDK-based “100% polling” dataplane workloads, such as a vCMTS Data Over Cable Service Interface Specification (DOCSIS) MAC dataplane. Key to their suitability for such workloads is the new User Wait (or WAITPKG) instruction set [2]. This instruction set can be used to place CPU cores into the new C-states, for example, when network load is below peak rates. Because they do not require root privilege, the instructions can be called directly from a user-space application.

The recommended instruction from this set is TPAUSE, which instructs the core to stop instruction execution for a defined period. During this period, the core can either enter the C0.1 or C0.2 states or switch to a hyper-thread sibling. The TPAUSE instruction is recommended as it can be used in several use-cases – amongst others, it can be used in:

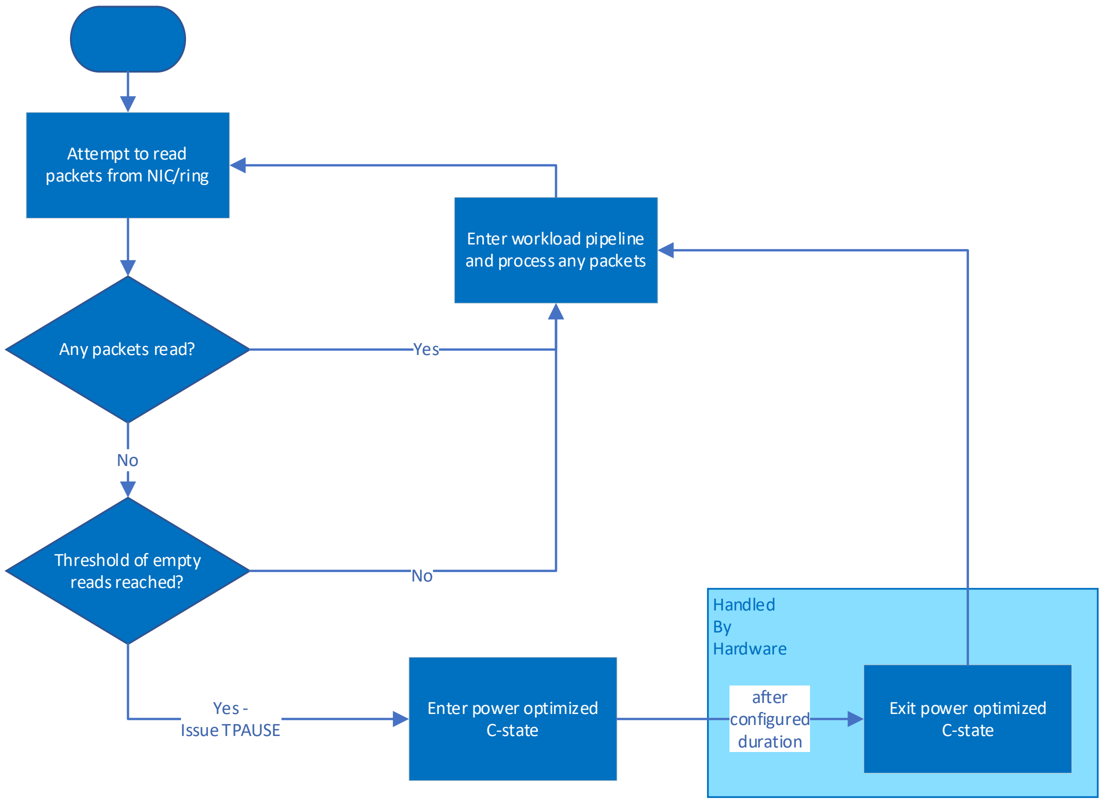
- scenarios which require multiple receive queue interfaces to be polled.
- scenarios which require time-critical processing e.g., messages which must be processed within a specific time window.

2.2 DPDK Enablement for Ethernet NIC Interfaces

The Ethernet Poll Mode Driver (PMD) Power Management [3] feature was added to DPDK v21.02 in the form of new APIs, to allow applications to enable and disable power savings on a per NIC port/queue basis. The API enables/disables an algorithm that tracks the rate at which packets are received when the NIC interface is polled and instructs an associated CPU core, through the User Wait instruction set, to enter the C0.1 or C0.2 states if they are available.

This API introduced several modes of operation, but the recommended mode for a vCMTS dataplane is the ‘Pause’ mode. When this mode is configured at application initialization time, the algorithm shown in Figure 1 below is enabled within the DPDK ‘ethdev’ and ‘power’ libraries, and within the DPDK PMD responsible for polling the NIC queue. The algorithm tracks the number of packets received on every read of a queue. Once a configured threshold of empty reads is reached, the PMD issues the TPAUSE instruction to place the core into the optimized power state for a configured number of microseconds. The core is re-activated (i.e. it exits the C-state) after the configured TPAUSE duration has elapsed.

Figure 1. User Wait ‘Pause’ Algorithm Implemented by DPDK


Attempt to read packets from NIC/ring
Enter workload pipeline and process any packets
Any packets read?
Yes
No
Threshold of empty reads reached?
No
Yes - Issue TPAUSE
Enter power optimized C-state
Handled By Hardware
after configured duration
Exit power optimized C-state

The following sections describe the API calls that can be used to enable, disable, and configure this algorithm. Recommended settings for configurable parameters of the algorithm are specified later in section 2.4.

Additional information on each of the APIs can be found in the DPDK Power Management documentation [3]. Examples of calling the APIs can also be found in the DPDK l3fwd-power sample application [4] or in the Intel® vCMTS Reference Dataplane [5].

2.2.1 Enabling PMD Power Management on a NIC Queue

Table 3. API to Enable PMD Power Management on a NIC Queue

API	

`__rte_experimental int rte_power_ethdev_pmgmt_queue_enable(`
`             unsigned int lcore_id,`
`             uint16_t port_id,`
`             uint16_t queue_id,`
`             enum rte_power_pmd_mgmt_type mode)`	

DPDK Version Introduced	
v21.02	
DPDK Header File	
`lib/power/rte_power_pmd_mgmt.h`	
Description	
This API call enables the PMD Power Management User Wait algorithm on a particular NIC Rx queue and specifies the power saving mechanism that will be used on its associated polling core. This API should be called during application initialization for each NIC Rx queue and core that requires power management. It is the only API call that is strictly required to allow a DPDK-based vCMTS application leverage power savings via the new power-optimized C-states for NIC interfaces. Once enabled, after a certain number of empty reads are detected on the NIC queue identified by `port_id`/`queue_id`, the CPU core identified by the `lcore_id` parameter will enter the C-state. This API call MUST be called after configuring the NIC port and queue. This API call MUST be called before starting the NIC port and queue.	
Parameters	
`lcore_id`	Logical-core identifier of the CPU core that polls and reads the NIC port/queue
`port_id`	Port identifier of the NIC port
`queue_id`	Queue identifier of the NIC port's queue
`mode`	The PMD Power Management mode to use: `RTE_POWER_MGMT_TYPE_PAUSE`

2.2.2 Disabling PMD Power Management on a NIC Queue

Table 4. API to Disable PMD Power Management on a NIC Queue

API

`__rte_experimental int rte_power_ethdev_pmgmt_queue_disable(`
`             unsigned int lcore_id,`
`             uint16_t port_id,`
`             uint16_t queue_id)`

DPDK Version Introduced
v21.02
DPDK Header File
`lib/power/rte_power_pmd_mgmt.h`
Description

When exiting the DPDK-based vCMTS application, the following API should be called for each of the NIC ports/queues that had PMD Power Management enabled during application initialization. This removes the PMD Power Management link between the NIC queue identified by `port_id`/`queue_id` and the CPU core identified by `lcore_id`. This API call MUST be called after stopping the NIC port and queue.	
Parameters	
`lcore_id`	Logical-core identifier of the CPU core that polls and reads the NIC port/queue
`port_id`	Port identifier of the NIC port
`queue_id`	Queue identifier of the NIC queue

2.2.3 Configuring PMD Power Management Settings on a NIC Queue

Additional functionality was added to the PMD Power Management feature in DPDK v22.07, to give the user the option to manipulate settings within the heuristics that apply the power management.

2.2.3.1 Configuring Maximum Empty Poll Parameter

Table 5. API to Configure Maximum Empty Poll Parameter on a NIC Queue

API	

`__rte_experimental void rte_power_pmd_mgmt_set_emptypoll_max(unsigned int max)`
`__rte_experimental unsigned int rte_power_pmd_mgmt_get_emptypoll_max(void)`	

DPDK Version Introduced	
v22.07	
DPDK Header File	
`lib/power/rte_power_pmd_mgmt.h`	
Description	
These APIs allow the user to set and get the maximum number of empty polls that must occur for a specific NIC queue before it's associated core enters the power saving state. Manipulating this setting gives additional control over the User Wait algorithm within the PMD Power Management feature, allowing for more aggressive or conservative power savings. The default maximum empty poll setting is 512, though it is recommended to reduce this as low as 32 for optimum power savings while still not impacting any key performance indicators (KPIs) of the vCMTS application. There is no strict requirement on when either of these APIs should be called.	
Parameters	
`max (set API only)`	The value to set the maximum number of empty pools to

2.2.3.2 Configuring Pause Duration

Table 6. API to Configure Pause Duration on a NIC Queue

API

`__rte_experimental int rte_power_pmd_mgmt_set_pause_duration(unsigned int duration)`
`__rte_experimental unsigned int rte_power_pmd_mgmt_get_pause_duration(void)`

DPDK Version Introduced
v22.07
DPDK Header File
`lib/power/rte_power_pmd_mgmt.h`

Description	
These APIs allow the user to set and get the duration (in microseconds) of the pause used by the 'Pause' mode of the PMD Power Management, and hence the maximum length of time the core is put into the power saving state for. The default pause duration is 1 microsecond. This API allows for manipulation of this duration. There is no strict requirement on when either of these APIs should be called.	
Parameters	
`duration (set API only)`	The value to set the pause duration to

2.3 DPDK Enablement for Ring Interfaces

Not all application threads will be based on receiving packets from an Ethernet NIC interface. It is very common to have threads which read packets from a DPDK ring interface. Such threads can also benefit from the User Wait power management techniques provided by the PMD Power Management feature.

Work is ongoing to add full support for the PMD Power Management techniques to DPDK ring interfaces – this will be available in a future DPDK version. In the meantime, a patch file has been implemented that mirrors some the PMD Power Management functionality for DPDK ring interfaces. The patch is available on the DPDK patchwork site at the link provided at reference [6]. There are 2 options available to apply the patch to DPDK v22.11:

1. Download the patch from a browser by using the link provided at reference [6], copy to the DPDK v22.11 repository clone and apply with the '`git am`' command.
2. Apply directly within the DPDK v22.11 repository clone by executing '`wget <patch link> -O - | git am`, replacing `<patch link>` with the URL provided at reference [6].

The patch implements the 'Pause' mode of the PMD Power Management feature and therefore follows the same algorithm shown previously in Figure 1.

The following sections describe the API calls that can be used to enable, disable and configure the algorithm. Recommended settings for configurable parameters of the algorithm are specified later in section 2.4.

2.3.1 Enabling PMD Power Management on a Ring

Table 7. API to Enable PMD Power Management on a Ring

API	

`__rte_experimental int rte_ring_pmgmt_ring_enable(`
`              unsigned int lcore_id,`
`              char *ring_name)`	

DPDK Version Introduced	
v22.11 (via patch)	
DPDK Header File	
`lib/ring/rte_ring_pmgmt.h`	
Description	
This API call enables the PMD Power Management 'Pause' mode algorithm on a particular ring and its associated polling core. This API should be called during application initialization for each ring and core that requires power management and is the only API call that is strictly required to allow a DPDK-based vCMTS application leverage power savings via the new power-optimized C-states for ring interfaces. Once enabled, after a certain number of empty reads are detected on the ring identified by `ring_name`, the CPU core identified by the `lcore_id` parameter will enter the C-state for a duration of 1 microsecond by default. This API call MUST be called after initializing the ring.	
Parameters	
`lcore_id`	Logical-core identifier of the CPU core that polls and reads the ring

`ring_name`	Name of the ring

2.3.2 Disabling PMD Power Management on a Ring

Table 8. API to Disable PMD Power Management on a Ring

API	

`__rte_experimental int rte_ring_pmgmt_ring_disable(`
`             unsigned int lcore_id,`
`             char *ring_name)`	

DPDK Version Introduced	
v22.11 (via patch)	
DPDK Header File	
`lib/ring/rte_ring_pmgmt.h`	
Description	
When exiting the DPDK-based vCMTS application, the following API should be called for each of the rings that had PMD Power Management 'Pause' mode enabled during application initialization. This removes the PMD Power Management link between the ring identified by `ring_name` and the CPU core identified by `lcore_id`. This API call MUST be called before freeing the ring.	
Parameters	
`lcore_id`	Logical-core identifier of the CPU core that polls and reads the NIC port/queue
`ring_name`	Name of the ring

2.3.3 Configuring PMD Power Management Settings on a Ring

2.3.3.1 Configuring Maximum Empty Poll Parameter

Table 9. API to Configure Maximum Empty Poll Parameter on a Ring

API	

`__rte_experimental void rte_ring_pmgmt_set_emptypoll_max(unsigned int max)`
 	
`__rte_experimental unsigned int rte_ring_pmgmt_get_emptypoll_max(void)`	

DPDK Version Introduced	
v22.11 (via patch)	
DPDK Header File	
`lib/ring/rte_ring_pmgmt.h`	
Description	
These APIs allow the user to set and get the maximum number of empty polls that must occur for a specific ring interface before it's associated core enters the power saving state. Manipulating this setting gives additional control over the User Wait algorithm within the PMD Power Management feature, allowing for more aggressive or conservative power savings. The default maximum empty poll setting is 512, though it is recommended to reduce this as low as 32 for optimum power savings while still not impacting any key performance indicators (KPIs) of the vCMTS application. There is no strict requirement on when either of these APIs should be called.	
Parameters	
`max (set API only)`	The value to set the maximum number of empty pools to

2.3.3.2 Configuring Pause Duration

Table 10. API to Configure Pause Duration on a Ring

API	

`__rte_experimental int rte_ring_pmgmt_set_pause_duration(unsigned int duration)`
`__rte_experimental unsigned int rte_ring_pmgmt_get_pause_duration(void)`	

DPDK Version Introduced	
v22.11 (via patch)	
DPDK Header File	
`lib/ring/rte_ring_pmgmt.h`	
Description	
These APIs allow the user to set and get the duration (in microseconds) of the pause used by the 'Pause' mode of the PMD Power Management. In 'Pause' mode, the default pause duration that the core is put into a power savings state for is 1 microsecond. The new API calls below allow for manipulation of this default duration. There is no strict requirement on when either of these APIs should be called.	
Parameters	
`duration (set API only)`	The value to set the pause duration to

2.4 Recommendations for vCMTS Integration

The preceding sections describe the API calls required to enable, disable, and configure use of the User Wait instruction set to optimize power consumption of a vCMTS deployment. However, it may still be unclear as to how exactly it can be enabled and configured within an application. This section aims to provide some additional tips and recommendations to leverage the feature for power savings in your vCMTS solution[1].

2.4.1 Hardware and Software Requirements

To make of use the User Wait instruction set and leverage power savings through the power-optimized C-states using the API calls described in the previous sections, certain criteria must be adhered to. The following are the basic hardware and software requirements that must be met:

- ✓ Must be deployed on 4th Gen Intel Xeon Scalable processor CPUs with User Wait instruction set support.
- ✓ Packet Processing pipeline must be based on DPDK v21.02 onwards, preferably DPDK v22.07 or later.
- ✓ Threads must be driven by reading packets with one of the following DPDK API calls:
    - ▪ rte_eth_rx_burst, rte_eth_rx_bulk, rte_ring_dequeue_burst, rte_ring_dequeue_bulk
- ✓ Ideally the threads should be designed in a run-to-completion manner, where once a 'burst' of packets is received they are processed and transmitted before the next burst is received.

2.4.2 Recommended Mode of Operation

As already described, the recommended mode to be used is 'Pause' mode, which can enabled for both NIC port/queue and DPDK ring interfaces by specifying the `RTE_POWER_MGMT_TYPE_PAUSE` option on the relevant enabling API call. The 'Pause' mode and the underlying TPAUSE instruction can be used in scenarios which require multiple receive interfaces to be polled, or where critical time-based processing is required. Both scenarios may be valid in a vCMTS application:

- A thread may need to poll both a NIC port/queue interface for IP packets and a ring interface for control plane or MAC management messages.
- A thread may need to poll a NIC port/queue interface, whilst also handling critical time-based operations, such as DOCSIS Upstream Bandwidth Allocation Map messages, at a defined fixed interval.

The 'Pause' mode ensures that core will only enter the C0 substate for the predefined duration before exiting and allowing other interfaces to be polled and/or other time-sensitive operations to be handled.

[1] The recommendations provided in this section are valid at the time of writing and are subject to change at any time and without notice.

2.4.3 Recommended Heuristics Settings

The default maximum empty poll parameter for the PMD Power Management modes is 512 empty polls before a C0 substate is triggered. This is a very conservative setting in terms of power management. It is recommended to reduce this setting using the relevant API calls to as low as 32 – this maximizes power savings by entering the C0 substates more frequently.

The default pause duration for the 'Pause' mode of the PMD Power Management feature is 1 microsecond. For dataplane threads it is not recommended to change this setting. For control-plane threads, increasing this duration using the relevant API calls may be considered, though likely not necessary.

The following table summarizes the default and recommended settings for each of the configurable parameters.

Table 11. Recommended PMD Power Management Settings

Configurable Parameter	Default Setting	Recommended Setting
Maximum Empty Polls	512	32
Pause Duration	1 microsecond	1 microsecond

2.4.4 Integrating User Wait without DPDK PMD Power Management Feature

Upon studying this documentation, it may become apparent that your vCMTS solution is not suitable to leverage the User Wait instruction set using the DPDK PMD Power Management feature in its current form. Alternatively, it may be desirable to leverage the User Wait instruction set for non-DPDK based software threads. Examples may include an upstream scheduler thread, control plane thread or other legacy threads not based on DPDK.

Table 12. API to Use User Wait without DPDK PMD Power Management

API	
`__rte_experimental int rte_power_pause(const uint64_t tsc_timestamp)`	
DPDK Version Introduced	
v20.11	
DPDK Header File	
`lib/eal/include/generic/rte_power_intrinsics.h`	
Description	
This API places the core on which it was ran into the power-optimized C-state for the duration specified by `tsc_timestamp`, through the User Wait TPAUSE instruction if it is available on the platform. The API can be called from any user-space application based on any criteria the developer deems worthy. The function can be called directly from a DPDK based application or the function can be studied and re-implemented for use in a non-DPDK based application as required.	
Parameters	
`tsc_timestamp`	The maximum duration in timestamp counter (TSC) ticks to pause the core for

2.5 Monitoring Usage

To monitor usage of the C0 substates by each CPU core, event counters are available. The `CPU_CLK_UNHALTED.C01` and `CPU_CLK_UNHALTED.C02` event counters represent the number of clock cycles that a core spends in the C0.1 and C0.2 state respectively. These counters can be monitored using the 'perf' tool from the Linux command line. C0.1 is represented by the `r10ec` event and C0.2 is represented by the `r20ec` event within the perf stat tool.

The counters increment for each clock tick of the CPU core that is spent in the power-optimized state. Actual percentage of time can then be computed by comparing the number of clock ticks to the TSC of the CPU. The 'perf' tool supports many command line options. The example below specifies both C0.1 and C0.2 events to be recorded for CPUs 2, 3, 66 and 67 at an interval of 1000 milliseconds. The reported number of ticks under the 'counts' column is a sum of all the CPUs specified in the command. A per-core utilization breakdown can be obtained by adding the '--per-core' option to the command. Additional options can be viewed using the '--help' option.

```
# perf stat -e r20ec -e r10ec -I 1000 -C 2,3,66,67
#           time             counts unit events
     1.001087032      3,555,349,439      r20ec
     1.001087032                  0      r10ec
     2.002228997      3,479,781,677      r20ec
     2.002228997                  0      r10ec
     3.003305175      3,479,237,965      r20ec
     3.003305175                  0      r10ec
```

There is also a counter to show whether the core is using C0.1 and C0.2, as opposed to just executing a pause instruction (a pause instruction is used when C0.1/C0.2 are not available on the platform). The counter is `CPU_CLK_UNHALTED.C0_WAIT` and it is incremented for every clock tick that the CPU core is using the C0 substates. It is represented by the `r70ec` event with the perf stat tool and can be viewed as follows.

```
# perf stat -e r70ec -I 1000 -C 2,3,66,67
#           time             counts unit events
     1.001092348      3,939,902,087      r70ec
     2.002250619      3,916,458,216      r70ec
     3.003327327      3,916,409,293      r70ec
```

This is good way of verifying that C0 substates are supported by the CPU and being used as expected by the CPU cores. On a platform where C0 substates are not supported and a pause instruction is executed instead, this r70ec event counter would always read 0.

3 P-states

3.1 Technology Overview

P-states refer to the specific frequency and voltage that a core operates at while executing instructions. They offer power savings by reducing the voltage and frequency of CPU cores while they run. Unlike with C-states, the execution of instructions continues as P-states are altered on the CPU core, albeit at an adjusted rate. This has made them an effective mechanism for reducing power consumption across all generations of Intel Xeon Scalable processors. P-states are beneficial to vCMTS deployments as reducing the operating voltage and frequency of the CPU cores that vCMTS dataplane threads are running on results in a corresponding reduction in power consumed by the CPU. Their suitability to vCMTS deployments is further strengthened by the significant reductions in P-state transition latency seen on both 3rd and 4th Gen Intel Xeon Scalable processors.

3.2 Configuring BIOS Settings

It is recommended that the following P-state specific settings be configured in the BIOS settings, to allow full control of the CPU P-states from user-space applications and tools. Note that the BIOS option names and menu locations listed in the table below are presented as a guide only. The exact option names and locations will vary depending on the server. Please contact your Intel® representative if you cannot find a particular option.

Table 13. P-state Specific BIOS Settings

BIOS Setup Menu	BIOS Option	Setting
Advanced → Power & Performance	CPU Power and Performance profile	Balanced Performance
Advanced → Power & Performance → CPU P State Control	Hardware P-states (HWP)	Disabled
Advanced → Power & Performance → CPU P State Control	Intel® Turbo Boost Technology	Disabled[2]
Advanced → Power & Performance → CPU P State Control	Energy Efficient Turbo	Disabled
Advanced → Power & Performance → CPU P State Control	Enhanced Intel® Speed Select Technology	Enabled

[2] While enabling and using Intel Turbo Boost Technology on CPU cores will improve vCMTS dataplane throughput, it will also increase power consumption. As this guide aims to reduce power consumption, it is therefore recommended to disable this option.

3.3 Selecting P-state Kernel Driver

The Linux kernel provides two different drivers to implement processor frequency scaling, namely the acpi-cpufreq and intel_pstate and drivers. Both drivers use the cpufreq subsystem of the Linux kernel to scale CPU frequencies up and down, but have their own different implementations and unique features:

- acpi-cpufreq is the older of the two drivers, and uses the governors provided by the cpufreq subsystem, where each governor implements a different performance scaling algorithm – these governors are: 'performance', 'powersave', 'userspace', 'schedutil', 'ondemand' and 'conservative'. For more details on these governors and the cpufreq subsystem as a whole, please see [7].
- intel_pstate, on the other hand doesn't use the cpufreq subsystem's governors. Instead, it implements its own 'performance' and 'powersave' policies internally. It is worth noting that while the 'performance' algorithm implemented by intel_pstate is similar to that of cpufreq, the 'powersave' algorithm of intel_pstate is more similar to the 'schedutil' governor of cpufreq. For more details on the intel_pstate driver, please see [8].

Both kernel drivers can be effectively utilized in a vCMTS deployment to scale CPU core frequencies, using the governors or policies provided and within configurable maximum and minimum frequency limits. However, it is recommended to use the acpi-cpufreq driver as it provides a 'userspace' governor, which allows userspace applications or tools to have full control of setting the CPU frequency. The remainder of this document assumes use of the acpi-cpufreq driver.

By default, the intel_pstate driver is used on Intel Xeon Scalable processors. In order to switch to the acpi-cpufreq driver, the following must be added to the kernel parameters (e.g. through the grub settings):

```
intel_pstate=disable
```

3.4 Manipulating P-states of CPU Cores

P-states settings can be manipulated from the Linux command line using the power.py python tool available from the Intel CommsPowerManagement GitHub repository [9]. Regardless of the kernel driver selected, this tool allows setting and getting of P-state settings such as the governor (or policy), and maximum, minimum, and configured frequencies on a per-core basis.

The usage is described in detail in the power.md file in the above repository.

3.4.1 Determining Available P-state Settings

Once downloaded to the server, the available P-state settings can be determined by running the tool as follows:

```
# ./power.py -i

     P-State Driver: acpi-cpufreq
 CPU Base Frequency: 0MHz
 Available P-States: [1801.0, 1800.0, 1700.0, 1600.0, 1500.0, 1400.0, 1300.0, 1200.0, 1100.0,
1000.0, 900.0, 800.0]
    Turbo Available: Yes (use pstate '1801.0')
     Number of CPUs: 128
Available Governors: ['conservative', 'ondemand', 'userspace', 'powersave', 'performance',
'schedutil']
 Available C-States: ['C1_ACPI', 'C2_ACPI', 'POLL']
```

The current P-state settings can be determined by running the tool as follows:

```
# ./power.py -l

==== ====== ====== ====== =========== ======= ======= =======
              P-STATE INFO                  C-STATES DISABLED?
Core    Max    Min    Now    Governor    POLL C1_ACPI C2_ACPI
==== ====== ====== ====== =========== ======= ======= =======
   0   1800   1800   1800   userspace      no      no      no
   1   1800   1800   1800   userspace      no      no      no
...
...
 126   1800   1800   1800   userspace      no      no      no
 127   1800   1800   1800   userspace      no      no      no
```

Depending on the CPU of the host platform, the displayed output will be some variation of what is shown above.

3.4.2 Configuring P-state Settings

Specific P-states can be configured on a per CPU core basis by running the tool as follows. This command specifies the 'userspace' governor (-g) with a minimum (-m), maximum (-M) and configured (-s) P-state of 1800 MHz on CPUs 0-127 (-r).

```
# ./power.py -g userspace -m 1800 -M 1800 -s 1800 -r 0-127
```

Setting the governor to 'userspace' and using the same value for the minimum, maximum and configured P-state setting effectively fixes the core frequencies to that frequency, 1800 MHz in above case, with no further adjustments being made by the governor algorithm.

To see a description of all possible options run:

```
# ./power.py -h
```

3.4.3 Verify P-state Settings

Once the P-state settings have been re-configured the changes can be verified either using the power.py tool or using the Linux turbostat utility from the command line.

```
# ./power.py -l
```

OR

```
# turbostat -d -i 1 -c 0-127
```

3.5 Recommendations for vCMTS Integration

The preceding sections describe how to manipulate P-states of CPU cores. However, it may still be unclear as to how exactly they can be used in a vCMTS deployment. This section aims to provide some practical advice for manipulating P-states in order to reduce power consumption[3].

3.5.1 Hardware and Software Requirements

There are very little hardware and software requirements of a vCMTS solution which can leverage P-states:

- ✓ Should be deployed on 3rd or 4th Gen Intel Xeon Scalable processor CPUs with P-states enabled.
    - Will also work on earlier generations, but P-state transition latency is higher.

3.5.2 Techniques for P-state Manipulation

The main challenge to optimize power usage with P-states is identifying opportunities where network load is lower than the maximum load for which the platform has been provisioned and subsequently altering the P-states accordingly. Traditional P-state controls within both the OS and hardware itself are unable to distinguish between low and high network activity on CPU cores processing packets using DPDK PMDs, which always show 100% CPU utilization since the PMDs relentlessly poll NIC or ring interfaces regardless of network load. As a result, alternative techniques are required to determine an accurate representation of the true CPU load the network traffic is generating.

3.5.2.1 Time-of-Day Based

The most basic technique to achieve power savings with P-states is to configure a pre-determined core frequency capable of handling the expected network load based on the largely predictable nature of DOCSIS traffic over a 24-hour period. Pre-adjusting the frequency of cores (e.g. using the power.py script mentioned previously) in such a manner is sure to provide power savings, particularly at nighttime when networks are usually much under-utilized. This approach, however, comes with some pitfalls. Pre-configuring the P-state leaves the operator susceptible to unexpected increases in network load atypical of a normal 24-hour period. The lack of real-time metrics used in this P-state tuning technique gives it clear limitations.

[3] The recommendations provided in this section are valid at the time of writing and are subject to change at any time and without notice.

3.5.2.2 Real-Time Telemetry Based

Additional software can be developed to make P-state manipulation decisions based on real-time telemetry from the underlying vCMTS application. Telemetry from the application including current throughput, number of empty polls occurring (i.e. the statistic that is used in the User Wait heuristics) or other metrics can all be used to accurately calculate the true CPU load of the core running the vCMTS application and the P-states can be manipulated accordingly (e.g. using the power.py script mentioned previously). With the correct implementation, such a technique not only optimizes power consumption based on network load but is also capable of reacting to unexpected changes in network load, thus ensuring KPIs are maintained.

Intel has developed the Intel® Infrastructure Power Manager (IPM) [10] to manipulate P-states based on real-time busyness telemetry from a DPDK-based vCMTS application. More info will be added to this document about Intel IPM in a future revision, but in the meantime, please contact your Intel representative to find out more about it.

3.6 Monitoring Usage

P-states can be monitored using the power.py and turbostat tools from the Linux command line.

```
# ./power.py -l
```

OR

```
# turbostat -d -i 1 -c 0-127
```

P-state statistics are also captured and available on a per CPU core basis under the following directory on a Linux OS.

```
# tree /sys/devices/system/cpu/cpu0/cpufreq/stats/
/sys/devices/system/cpu/cpu0/cpufreq/stats/
├── reset
├── time_in_state
├── total_trans
└── trans_table
```

4 Summary

Optimizing the power consumption of vCMTS deployments, and in particular the DOCSIS MAC dataplane, is becoming more and more important. Intel Xeon Scalable processors provide several techniques to achieve this goal.

The document has shown how the Intel User Wait instruction set can be integrated into a vCMTS deployment to optimize power consumption based on real-time network load. Use of these instructions allows CPU cores enter new power-optimized C-states which are available on 4th Gen Intel Xeon Scalable processors. The document covered all the relevant API calls and recommended settings for enabling this feature within DPDK. Supporting documentation in Table 2 provides further information, including the power savings achievable in a vCMTS deployment on such processors using this technique.

The document has also shown how to manipulate P-states of CPU cores and contains some suitable techniques for leveraging P-states to further optimize power consumption in a vCMTS deployment on 4th Intel Xeon Scalable processors, and to enable power efficiencies on previous 1st, 2nd, and 3rd Gen Intel Xeon Scalable processors.