Software Development Kit User Guide

Contents of this document are subject to change without notice.

November 2013

PUBLISHED BY
Cavium, Inc.
2315 North 1st Street
San Jose, CA 95131
Phone: 650-623-7000
Fax: 650-625-9751
Email: sales@cavium.com
Web: http://www.cavium.com

Revision History

Release	Date	Changes
3.1.0	November 2013	Replaced bootoct and oct-shutdown-app utilities for controlling SE applications from Linux with oct-app-ctl. Removed Valgrind sections. Added -perf3 and --perf4 command-line options to oct-profile, oct-remote-profile, and oct-top (only available in OCTEON III); updated L2 events for oct-remote-profile.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Cavium OCTEON Software Developer Kit (SDK) includes a set of analysis tools for Linux and Simple Executive applications. Most tools run either natively on an OCTEON target running Linux or on a remote host connected to an OCTEON board.

Native tools are provided with both the embedded root file system and Debian. The tools are usually hosted in the `/usr/bin` directory.

This guide provides instructions for using the tools, organized in the following sections:

- The oct-profile Tool
- The oct-remote-profile Tool
- The oct-top Tool
- The otrace Tool
- The perf Tool
- The perfzilla Utility
- SE-Dump and the oct-remote-coredump Tool
- The viewzilla Tool
- Controlling SE Applications from Linux

Additional examples of output are provided in Examples.

Chapter 2

The oct-profile Tool

2.1 Introduction

The oct-profile utility is a profiling tool for OCTEON. It is similar to oct-remote-profile, with the key difference that oct-profile can resolve all PC values to respective routines. This facilitates the display of the results with the relevant information of routines and source code line number. An `ELF` file is required for profiling with oct-profile.

2.2 Features

The oct-profile utility includes the following features:

- Displays routine names and sample counts on respective cores.
- Displays PC (Program Count) values and associated routine information.
- Shows the source file and line number associated with PC value.
- Supports per-core view.
- Handles up to four core performance counter events.
- Handles up to four L2 (Level-2 cache) performance counters.
- Controls output refresh rate.

2.3 Command-line Options

The supported command-line options for `oct-profile` are listed in the following table:

`oct-profile [--line] [--address] <elf>"`

Table 2–1 oct-profile Command-line Options

Option	Description
`<ELF file>`	Mips ELF file from which debugging information is loaded.
`--line`	Display line number information.
`--address`	Display exact PC address.
`--perf1=<Core counter>`	Set up Core performance counter 1.
`--perf2=<Core counter>`	Set up Core performance counter 2.
`--perf3=<Core counter>`	Set up Core performance counter 3. **Note:** Only available in OCTEON III (CN7XXX and later).
`--perf4=<Core counter>`	Set up Core performance counter 4. **Note:** Only available in OCTEON III (CN7XXX and later).
`--l2perf1=<L2 counter>`	Set up L2 performance counter 1.

Option	Description
`--l2perf2=<L2 counter>`	Set up L2 performance counter 2.
`--l2perf3=<L2 counter>`	Set up L2 performance counter 3.
`--l2perf4=<L2 counter>`	Set up L2 performance counter 4.
`--dwell=<seconds>`	Number of seconds to refresh display.
`--core_mask=<mask>`	Mask that specifies the cores to sample.

2.4 Getting Started

The profiler can be invoked to profile applications running natively on OCTEON or applications running on an x86 development host targeting an OCTEON PCI/PCIe target board.

Note: The utility must always be invoked with the `ELF` file for the application running over PCI; it does not support profiling applications running natively on OCTEON.

The utility profiles `traffic-gen` to execute the respective commands.

Table 2–2 oct-profile Command Quick Reference

Command	Description
`oct-profile /tftpboot /traffic-gen`	Resolve raw PC values output to routines during runtime.
`oct-profile –line /tftpboot/traffic-gen`	Display the routines for corresponding PC values along with source code and line numbers.
`oct-profile –address /tftpboot/traffic-gen`	Display the routines for corresponding PC values along with associates PC values for that routine.
`oct-profile –perf1=dissue /tftpboot/traffic-gen`	Program the perf counter 1 for dissue event and display the results.
`oct-profile –dwell=2 /tftpboot/traffic-gen`	Wait for 2 seconds before displaying the next set of sample information.
`oct-profile –core_mask=0x23 /tftpboot/traffic-gen`	Profile cores 0,1 and 5.

Note:

- Post profiling tools cannot be run on the results of `oct-profile`.
- The results of `oct-profile` cannot be used as inputs for perfzilla.

2.5 Understanding Output

When `oct-profile` is invoked from an x86 host with OCTEON connected over PCI running the `traffic-gen` Simple Executive application, the output displays as shown below.

```
target # oct-profile
core#: 0    1    2    3    4    5     _sprintf_r
       1    0    0    0    0    0     _vfprintf_r
       1    0    0    0    0    0     cvmx_sysinfo_get
       1    0    0    0    0    0     fdt_offset_ptr
       2415 0    0    0    0    0     main
       2415 0    0    0    0    0     statistics_gatherer
       0    3251 3251 3251 3251 3251  packet_transmitter
       0    203  185  175  168  174   cvmx_get_cycle
       0    203  185  175  168  174   cvmx_clock_get_count
       0    91   69   62   76   87    cvmx_pow_work_request_async_nocheck
       0    20   12   6    19   16    cvmx_send_single
       0    20   12   6    19   16    cvmx_write64_uint64
       0    97   120  103  99   114   cvmx_scratch_write64
       327  0    0    0    0    0     periodic_update
       14   0    0    0    0    0     cvmx_get_cycle
       14   0    0    0    0    0     cvmx_clock_get_count
       505  0    0    0    0    0     uart_printf
```

The results display the sample count on respective cores for a particular routine. In the background, oct-profile retrieves the PC values and associates sample hits. It resolves all PC values to the associated routines. This makes the results more understandable than those returned by running `oct-remote-profile` or `oct-linux-profile`.

Chapter 3

The oct-remote-profile Tool

3.1 Introduction

Profiling provides a way to analyze and fix performance issues on a system. Code running on OCTEON can be profiled using oct-remote-profile.

The oct-remote-tool can run natively on OCTEON, or it can run on an x86 host targeting an OCTEON PCI/PCIe target board. The tool is invoked using a wrapper, depending on which method you are using:

- `oct-linux-profile`—the equivalent of `OCTEON_REMOTE_PROTOCOL=linux oct-remote-profile`. Use when running the tool natively on OCTEON.
- `oct-pci-profile`—the equivalent of `OCTEON_REMOTE_PROTOCOL=pci oct-remote-profile`. Use when running the tool on an x86 host targeting an OCTEON PCI/PCIe target board.

The oct-remote-profile utility supports both continuous statistical profiling and event-based profiling when run on an x86 host, while only statistical profiling is supported when running natively on OCTEON.

- **Statistical Profiler—**Profiles are collected by continuously sampling the registers on each core. The profiler collects PC values and resolves them to symbols and lines of code (LOC) in source code using post-profiling tools like perfzilla.
- **Event-based Profiler—**Event-based profiling takes advantage of dedicated performance counters capable of measuring events at the hardware level on each core. These counters increment once per each event and can generate interrupt on counter overflow. The oct-remote-profile utility can use these interrupts to generate samples, which are then mapped by post-profiling tools; for example, PC values to source code routine and line numbers.

3.2 Features

The oct-remote-profile utility includes the following features:

- Performs statistical profiling of Linux and Simple Executive applications.
- Performs interrupt-based profiling of a Simple Executive application. For similar profiling on Linux, refer to the Oprofile section (*Performance Profiling Using Oprofile*) of the documentation found under `doc/html/index.html`.
- Displays OCTEON hardware information such as memory, FPA (floating point arithmetic), and POW.
- Profiles an application on a subset of cores.
- Collects profile information specific to a core.
- Controls the profile collection rate.

- Profiles the Simple Executive application for a specific hardware event.

  Note: Currently, OCTEON and OCTEON Plus only support profiling a single event.

3.3 Command-line Options

The supported command-line options for oct-remote-profile are listed in the following table:

Table 3–1 `oct-remote-profile` Command-line Options

Option	Description
`--perf1`	Core counter
`--perf2`	Core counter
`--perf3`	Core counter **Note:** Only available in OCTEON III (CN7XXX and later).
`--perf4`	Core counter **Note:** Only available in OCTEON III (CN7XXX and later).
`--l2perf1`	L2 counter
`--l2perf2`	L2 counter
`--l2perf3`	L2 counter
`--l2perf4`	L2 counter
`--list-events`	List available events.
`--events`	`<event name>:<count>` Enable profiling based on event and threshold count.
`--fast`	Sample as quickly as possible.
`--samples`	`<samples per second>` Number of samples taken per second. **Note:** High sample rates may not yield expected results.
`--dwell`	`<seconds>` Number of seconds between printing.
`--perfonly`	Only display performance counters.
`--count`	`<number>` Number of dwell times to sample and then exit.
`--core_mask`	`<mask>` Specify cores to mask.

The following table lists the options available for `perf1` and `perf2` events (OCTEON II):

Table 3–2 OCTEON II –`perf1`, `--perf2` Events

`clk`	`issue`	`ret`	`nissue`
`sissue`	`dissue`	`ifi`	`br`
`brmis`	`j`	`jmis`	`replay`
`iuna`	`trap`	`uuload`	`uustore`
`uload`	`ustore`	`ec`	`mc`
`cc`	`csrc`	`cfetch`	`cpref`
`ica`	`ii`	`ip`	`cimiss`
`wbuf`	`wdat`	`wbufld`	`wbuffl`
`wbuftr`	`badd`	`baddl2`	`bfill`
`ddids`	`idids`	`didna`	`lds`

lmlds	iolds	dmlds	sts
lmsts	iosts	iobdma	dtlb
dtlbad	itlb	sync	synciob
syncw	eretmis	likmis	hazard-trap

The following table lists the options available for `perf1`, `perf2`, `perf3`, and `perf4` events (OCTEON III):

Table 3–3 OCTEON III `--perf1`, `--perf2`, `--perf3`, `--perf4` Events

none	clk	issue	ret
nissue	sissue	dissue	ifi
br	brmis	j	jmis
replay	iuna	trap	uuload
uustore	uload	ustore	ec
mc	cc	csrc	cfetch
cpref	ica	ii	ip
cimiss	wbuf	wdat	wbufld
wbuffl	wbuftr	badd	baddl2
bfill	ddids	idids	didna
lds	lmlds	iolds	dmlds
sts	lmsts	iosts	iobdma
dtlb	dtlbad	itlb	sync
synciob	syncw	dutlb	iutlb
cdmiss	eretmis	likmis	hazard-trap
fpunimp-trap	fphazard-trap	fparith-exc	fpmovc1
fpcopyc1	fpcompare	fpbranch	fpmov
fpabs	fpadd	fpcvt	fpmul
fpmadd	fpdiv	fpsqrt	fpload
fpstore	fpall		

The following table lists the events available for Level 2 cache counters in OCTEON II:

Table 3–4 OCTEON II L2 Counter Events

none	hit	miss
no-alloc	victim	sc-fail
sc-pass	lfb-valid	lfb-wait-lfb
lfb-wait-vab	quad0-index-bus	quad0-read-bus
quad0-banks-inuse	quad0-wdat-inuse	quad1-index-bus
quad1-read-bus	quad1-banks-inuse	quad1-wdat-inuse
quad2-index-bus	quad2-read-bus	quad2-banks-inuse
quad2-wdat-inuse	quad3-index-bus	quad3-read-bus
quad3-banks-inuse	quad3-wdat-inuse	

The following table lists the events available for Level 2 cache counters in OCTEON III:

Table 3–5 OCTEON III L2 Counter Events

none	hit	miss	no-alloc
victim	sc-fail	sc-pass	lfb-valid
lfb-wait-lfb	lfb-wait-vab	quad0-index	quad0-read
quad0-bank	qu0-wdat	quad1-index	quad1-read
quad1-bank	quad1-wdat	quad2-index	quad2-read
quad2-bank	quad2-wdat	quad3-index	quad3-read
quad3-bank	quad3-wdat	quad4-index	quad4-read
quad4-bank	quad4-wdat	quad5-index	quad5-read
quad5-bank	quad5-wdat	quad6-index	quad6-read
quad6-bank	quad6-wdat	quad7-index	quad7-read
quad7-bank	quad7-wdat		

3.4 Getting Started

The profiler can be invoked to profile applications running natively on OCTEON or running on an x86 development host targeting OCTEON processors over PCI.

3.4.1 Running Natively on OCTEON

You can invoke `oct-linux-profile` to profile any application running on OCTEON. This includes any Linux application with an embedded root file system and Debian.

Table 3–6 oct-remote-profile Command Quick Reference - Native OCTEON

Command	Description
`oct-linux-profile`	Profile the code and generate PC values along with sample count for all cores.
`oct-linux-profile --perfonly`	Display only perf counters and OCTEON statistics such as TADs, DRAM memory, FPA pool memory, SSO Input/output queues.
`oct-linux-profile --perf1=dissue`	Set the perfcounter1 to `dissue` event and profile the application code. The output displays perfcounter1 values for the `dissue` event. This is a statistical profiler.
`oct-linux-profile --count=10`	Sample 10 times and exit the profiler.
`oct-linux-profile –core_mask=0x23`	Profile cores 0, 1, and 5.
`oct-linux-profile --dwell=2`	Wait two seconds before printing next set of sample data.
`oct-linux-profile --fast`	Get samples as fast as possible. Max samples are approximately 1000/sec.

3.4.2 Running on OCTEON Connected over PCI

Invoke `oct-pci-profile` from an x86 development host to profile applications running on OCTEON over PCI. This configuration supports both Linux and Simple Executive applications.

Table 3–7 oct-remote-profile Command Quick Reference - OCTEON Connected over PCI

Command	Description
`oct-pci-profile`	Profiler runs over PCI and can profile Linux or SE application.
`oct-pci-profile --perfonly`	Collect the perf counters and OCTEON statistics such as TADs, DRAM memory, FPA pool memory, SSO Input/output queues over PCI.
`oct-pci-profile --perf1=dissue`	Set perfcounter1 to dissue event and profile the application code over PCI. The output displays perfcounter1 values for the dissue event. This is a statistical profiler.
`oct-pci-profile --count=10`	Sample 10 times and exit the profiler.
`oct-pci-profile --core_mask=0x23`	Profile cores 0, 1, and 5.

Command	Description
`oct-pci-profile --dwell=2`	Wait two seconds before collecting next set of samples over PCI.
`oct-pci-profile --fast`	Get samples as fast as possible over PCI.
`oct-pci-profile –list-events`	List all events supported for Interrupt profiler.
`oct-pci-profile –events=cimiss:100000`	Invoke event-based profiler for cimiss event and the sample rate of 100000.

3.5 Understanding Output

When `oct-pci-profile` is invoked from an x86 host with OCTEON connected over PCI running the `traffic-gen` SE application, the output displays as shown below.

```
core#:                      0      1      2      3      4      5
0x0000000010004f78:         7      0      0      0      0      0
0x0000000010004f8c:         1      0      0      0      0      0
0x0000000010004fe8:         267    0      0      0      0      0
0x0000000010004fec:         119    0      0      0      0      0
0x0000000010004ff0:         109    0      0      0      0      0
0x0000000010004ff8:         1      0      0      0      0      0
0x0000000010005020:         3      0      0      0      0      0
0x0000000010005028:         1      0      0      0      0      0
0x0000000010009428:         0      104    95     97     110    85
0x0000000010009430:         0      67     63     131    76     65
0x0000000010009434:         0      232    216    369    247    249
0x0000000010009630:         0      68     88     98     80     80
0x0000000010009634:         0      63     84     87     69     80
0x000000001000963c:         0      210    230    210    212    245
0x0000000010009640:         0      70     70     67     64     78
0x0000000010009644:         0      211    246    214    237    204
0x000000001000964c:         0      80     78     59     114    98
0x0000000010009734:         0      218    208    193    244    226
0x000000001000973c:         0      89     74     107    95     91
0x0000000010009740:         0      12     7      18     11     16
0x0000000010009750:         0      53     47     47     36     55
0x0000000010009754:         0      19     16     10     14     21
0x0000000010009758:         0      16     17     13     10     13
0x0000000010009778:         0      221    229    189    213    240
0x000000001000977c:         0      233    251    194    203    186
0x0000000010009784:         0      88     103    81     62     63
0x0000000010009788:         0      262    238    257    234    231
0x0000000010009ea8:         0      84     97     71     73     78
0x0000000010009eac:         0      85     82     53     77     68
0x0000000010009eb4:         0      83     68     70     69     65
0x0000000010009eb8:         0      144    122    121    134    161
0x0000000010009ebc:         0      239    222    195    267    253
```

The results display PC values and total samples collected for the dwell period on all available cores (for example, six cores for CN63XX).

In the sample entry, 0x0000000010009644: 0 211 246 214 237 204:

- 0x0000000010009644 is the PC value

  On OCTEON CN63XX hardware, this had:
- 0 hits on 0th core
- 211 hits on 1st core
- 246 hits on 2nd core
- 214 hits on 3rd core
- 237 hits on 4th core
- 204 hits on 5th core.

Interrupt Profiler

For event-based profiling, the output depends on the events that would occur when profiling an application. If the sampling rate is too high, no events would occur and no samples would be collected. In such cases, oct-remote-profile displays an informative message such as the following:

```
oct-remote-profile --events=cimiss:100000000000

PCIE port 0

^C

INFO: No samples collected for 'cimiss' event with 100000000000 sample rate

INFO: Try with smaller sample-rate for same event
```

Chapter 4

The oct-top Tool

4.1 Introduction

The `oct-top` tool provides real-time information for applications running on an OCTEON target. The tool can run natively on OCTEON, or it can run on an x86 host targeting an OCTEON PCI/PCIe target board.

The `oct-top` tool generates continual reports about the state of OCTEON. The reports display the following information:

- Top CPU-consuming routines
- State of Tag and Data units (TADs)
- DRAM memory
- Floating Point Arithmetic (FPA) pool memory
- POW input/output queues

4.2 Features

The oct-top tool includes the following features:

- Prints sorted lists of frequently-called routines of applications running on an OCTEON target.
- Displays raw Program Counter (PC) values, sorted by sample count.
- Displays PC values and associated routines in sorted order.
- Shows the source file and line number associated with PC value.
- Provides per-core view.
- Handles up to four core performance counter events.
- Handles up to four L2 (Level-2 cache) performance counters.
- Controls screen refresh rate.

4.3 Command-line Options

The supported command-line options for oct-top are listed below; each option is described in the table that follows.

```
oct-top [--line] [--address] [--perf1=counter]
[--l2perf1=L2counter] [--dwell=<seconds>] [--core_mask=mask]
[--cpu=coreno] <elffile>
```

Table 4–1 oct-top Command-line Options

Option	Description
`<ELF file>`	Mips ELF file from which to load debugging information.
`--line`	Display line number information.
`--address`	Display exact PC address.
`--perf1=<Core counter>`	Set up core performance counter 1.
`--perf2=<Core counter>`	Set up core performance counter 2.
`--perf3=<Core counter>`	Set up core performance counter 3. **Note:** Only available in OCTEON III (CN7XXX and later).
`--perf4=<Core counter>`	Set up core performance counter 4. **Note:** Only available in OCTEON III (CN7XXX and later).
`--l2perf1=<L2 counter>`	Set up L2 performance counter 1. **Note:** Level-2 cache is shared by all cores and I/O components on Cavium OCTEON processors.
`--l2perf2=<L2 counter>`	Set up L2 performance counter 2.
`--l2perf3=<L2 counter>`	Set up L2 performance counter 3.
`--l2perf4=<L2 counter>`	Set up L2 performance counter 4.
`--dwell=<seconds>`	Number of seconds to refresh display.
`--core_mask=<mask>`	Mask that specifies the cores to sample.
`--cpu=<core no>`	Per-CPU `top` information, [0-n cores].
`-r`	Reverse the results of output.
`-s`	OCTEON Memory, Perf counter statistics.
`-k`	Use kernel symbols from `/proc/kallsyms` **Note:** This command is only supported when `oct-top` is run natively on OCTEON running Linux. The command can resolve the PC values to symbols (routine names), even in the absence of an `ELF` file. This option is not supported for Simple Executive applications.

4.4 Getting Started

You can invoke oct-top to run natively on OCTEON or on an x86 development host targeting the OCTEON processors over PCI. The utility can be invoked with or without an `ELF` file.

4.4.1 `oct-top` Commands Quick Reference

Table 4–2 oct-top Command Quick Reference

Command	Description
`oct-top`	Display PC values for all cores in ascending order
`oct-top traffic-gen`	Display functions which consume the most CPU time

Command	Description
`oct-top traffic-gen --line`	Display routines as well as the source code information corresponding to samples collected
`oct-top traffic-gen –address`	Display routines and associated PC values that fall under corresponding routines with sample count
`oct-top --dwell=2`	Refresh output every two seconds
`oct-top –core_mask=0x9 --cpu=3`	Display top information for core 3 where as limit the samples to coremask 0x9; that is, 0, 3 cores
`oct-top --cpu=4`	Display top information of cpu 4
`oct-top –r`	Display PC values for all cores in ascending order
`oct-top –s`	Display OCTEON statistics information such as TADs, DRAM memory, FPA pool memory, POW Input/output queues
`oct-top –k`	Use kernel symbols from `/proc/kallsyms`. Only supported when running natively on an OCTEON booted with Linux, where symbols can be resolved using `/proc/kallsyms` even in absence of an ELF file.

4.4.2 Running `oct-top` Natively on OCTEON

Follow the instructions below to invoke oct-top from Linux running natively on OCTEON. You can run oct-top with a Linux `ELF` image or on a Simple Executive application booted using the `oct-apt-ctl` utility.

1. **Enable python and `oct-top` support.**

   Because oct-top has a python front end, it requires python support included in the embedded root file system and Debian. By default python support is disabled; to enable support, set the following configuration options in the `default.config` file:

   ```
   CONFIG_python=y
   CONFIG_octeon-top=y
   ```

   You can enable the options by opening the configuration file in a text editor (for example, vi) and adding the options to the file:

   ```
   $ vi linux/embedded_rootfs/default.config
   ```

   or by running `make menuconfig` and selecting the options for `python` and `octeon-top` in the interface.

   ```
   $ cd linux/embedded_rootfs
   $ make menuconfig
   ```

2. **Build the kernel.**

   After setting the options to enable python and `oct-top` support, rebuild the kernel.

   ```
   $ make [kernel|kernel-deb]
   ```

4.4.3 Running OCTEON Over PCI

All commands listed in oct-top Commands Quick Reference are valid except `oct-top -k`, which is only supported when running oct-top natively on an OCTEON booted with Linux.

4.5 Understanding Output

The following sections describe output when running `oct-top` with different options.

`oct-top (with no options)`

When `oct-top` is invoked from an x86 host with OCTEON connected over PCI running the `traffic-gen` SE application, the output displays as shown below.

```
target# oct-top /tftpboot/traffic-gen
Function Name:                Total hits
fdt_next_tag:                    1
fdt_offset_ptr:                  1
memmove:                         2
cvmx_sysinfo_get:                4
cvmx_get_cycle:                 14
cvmx_clock_get_counta:          14
cvmx_send_single:               75
cvmx_write64_uint64:            75
periodic_update:               404
uart_printf:                   551
cvmx_scratch_write64:          604
cvmx_get_cycle:               1016
cvmx_clock_get_count:         1016
main:                         2613
statistics_gatherer:          2613
packet_transmitter:          17880
```

`oct-top` displays the function name and aciates samples collected for the routine during the dwell time (default is five seconds). Results are sorted in ascending order.

```
target# oct-top -s
Core Counters:
brmis 13104238[0], 12457620[1], 12186461[2], 12168803[3], 11557216[4], 275724[5],
total (excluding 0) brmis 48645824,
sync 39306[0], 39811[1], 39236[2], 38657[3], 39045[4], 26891[5],
total (excluding 0) sync 183640,
```

- Results display the default core performance counter events and counter value for dwell seconds.
- `brmis` (brand misdirects) and `sync` events are counted by default and can be changed using `–perf1` and `–perf2` options invoked with `oct-top`.
- `total` is the sum total of all performance events on available cores [0-maxcores].

L2 Statistics

L2 refers to the Level-2 cache, which is shared by all cores and I/O components on Cavium OCTEON processors. This can be bypassed using particular CMI transactions and can also be partitioned.

CN68XX has four TADs (Tag and Data units), each containing 1 MB of L2 cache; CN63XX has one TAD.

Table 4–3 L2 Statistics for TAD 0

```
bus_xmc(addr) count:          20
bus_xmd(store) count:         30
bus_rsc(commit) count:        18
bus_rsd(fill) count:          20
bus_ioc(IO req) count:        4
bus_ior(IO req) count:        1
```

```
hit count:                          191666339
miss count:                         17723607
lfb-wait-lfb bus utilization:       96%
lfb-wait-vab bus utilization:       0%
```

where:

Table 4–4 L2 Statistic Descriptions

Statistic	Description
`bus_xmc(addr) count:`	L2 cache ADD bus performance counter value
`bus_xmd(store) count:`	L2 cache STORE bus performance counter value
`bus_rsc(commit) count:`	L2 cache COMMIT bus performance counter value
`bus_rsd(fill) count:`	L2 cache FILL bus performance counter value
`bus_ioc(IO req) count:`	L2 cache IOC performance counter value
`bus_ior(IO req) count:`	L2 cache IOR performance counter value

The following display refers to default L2 performance counter events and can be changed using `–l2perf1`, `--l2perf2`, `--l2perf3`, `--l2perf4` options:

```
hit count:                          191666339
miss count:                         17723607
lfb-wait-lfb bus utilization:       96%
lfb-wait-vab bus utilization:       0%
```

where:

- **LFB** = Inflight address Buffers.
- **VAB** = Victim Address Buffers

DRAM Statistics

Includes the following statistics:

- `DRAM ops count`: This is a 64-bit Perf counter that increments each time the DDR3 databus is used.
- `dclk count`: 64-bit performance counter that increments every clock cycle.

```
DRAM ops count:    251089710
dclk count:        2667060265
utilization:       9.41%
```

FPA Pool Statistics

Displays FPA pool statistics such as free Packet Buffers, WQE buffers, and PKO cmd buffers on OCTEON.

```
FPA pool 0 (Packet Buffers):    263 free
FPA pool 1 (WQE buffers):       860 free
FPA pool 2 (PKO cmd buffers):   941 free, PKO output queues 128
```

POW Qos Input Queue Statistics

```
Queue 0:        WQE count              51   Buffers allocated       50
Queue 1:        WQE count              50   Buffers allocated       50
Queue 2:        WQE count              0    Buffers allocated       0
Queue 3:        WQE count              0    Buffers allocated       0
Queue 4:        WQE count              0    Buffers allocated       0
Queue 5:        WQE count              0    Buffers allocated       0
Queue 6:        WQE count              0    Buffers allocated       0
Queue 7:        WQE count              0    Buffers allocated       0
Total WQEs in all QOS levels:          103 (in-unit & in-memory)
Total Free Buffers in all QOS levels:  901
```

Chapter 5

The otrace Tool

5.1 Introduction

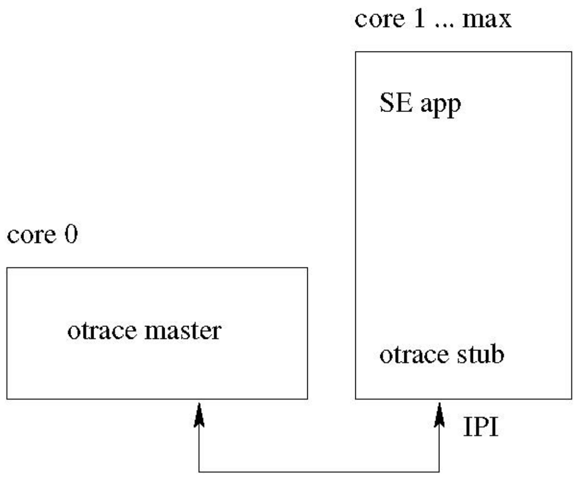
The OCTEON SE tracing tool `otrace` provides insight into a Simple Executive application's runtime behavior. The `otrace` tool has two parts:

- `stub`—Code invoked through inter-processor interrupts in the traced application.
- `otrace master`—An SE application itself that accepts user input and talks to the `stub`.

The figure below shows an example runtime setup with the `otrace master` running on one core and the application with the `stub` on the other cores.

Note: The `otrace master` does not have to run on core 0.

Figure 5–1 `otrace master` and `stub` for SE Application



The otrace tool was created with applications that process continuous workload in mind, for example, `passthrough`. The application and `otrace master` run concurrently during the tracing session while the user queries the application through the `otrace master` interactively. otrace does not explicitly support tracing applications that run to the end without waiting for user interaction (for example, `crypto`).

The following sections describe how to build an SE application to be traced and how to build a customized `otrace master` for the application. A list of `otrace master` commands is provided, as well as example tracing output.

5.2 Preparing to Trace

The following section provides instructions for setting up your SE application to be traced:

1. **Prepare an SE application for otrace.**

   Embed static probes in the functions with the GCC option `-finstrument-functions` in the application's `makefile`:

   For example:

   ```
   OCTEON_CFLAGS_GLOBAL_ADD += -finstrument-functions \
            -finstrument-functions-exclude-file-list= \
                             ${OCTEON_ROOT}/executive, \
                                 ${OCTEON_ROOT}/target
   ```

   In this example, the application's `makefile` specifies that all functions except those in the files under `executive/` and `target/` are to be instrumented.

   As another example:

   ```
   OCTEON_CFLAGS_GLOBAL_ADD += -finstrument-functions \
            -finstrument-functions-exclude-file-list= \
                                                 cvmx
   ```

   creates static probes for all functions except for those in files whose names contain `cvmx`.

   Note: For more ways to include/exclude files and functions for instrumentation, see gcc(1).

   A function instrumented in this manner has two probes: one at the entry and the other at the exit.

2. **Prepare the `otrace master` for the target SE application.**

   The `otrace master` must know the target application's symbols and instrumentation points (static probes) to provide a more user-friendly interface. Depending on how the `otrace master` is run, this information is acquired statically or dynamically.

   - **`otrace master` as an SE application**

     The symbol and instrumentation information is extracted at build time and embedded in the `otrace master` binary. This is accomplished by specifying the target while building the `otrace master.`

     For example:

     `TRACED_SE_APP_BIN=$(OCTEON_ROOT)/examples/passthrough/passthrough`

     In this example, `otrace master` `Makefile` designates the application `passthrough` as the target application.

     Note: In this case, both the `otrace master` and the target SE application are built with `OCTEON_TARGET=cvmx_64`.

   - **`otrace master` on Linux**

     When the `otrace master` runs on Linux as a userland program, there is no need or way to specify the target at build time. Instead, the `otrace master` provides the command `ldsyms se_app_ bin` to load the information at runtime from the target SE application binary `se_app_bin`.

Note: In this case, while the SE app is built with `OCTEON_TARGET=cvmx_64`, the `otrace master` is built with `OCTEON_TARGET=linux_64` to run on Linux.

Make sure that the target SE appliction is built with the symbol information.

You can now trace your application using The otrace master Commands.

5.3 The `otrace master` Commands

The `otrace master` runs on one core. It provides a simple Command-Line Interface (CLI) with `vi`-style editing through a UART console. The `otrace` commands are divided into three groups:

- System Commands
- Tracing Commands
- Miscellaneous Commands

5.3.1 System Commands

The System commands include:

Table 5–1 otrace System Commands

Option	Description
`exit`	Exits `otrace master.`
`help [cmd|group]`	Displays help information for a command or command group (tracing, system, or misc), including: • A list of available commands with short summaries in alphabetical order when invoked without a parameter. • A list of commands and their summaries in a specific group with tracing, system, or miscellaneous. • Detailed description of a command when `cmd` supplied as the parameter.
`ldsyms se_app_bin`	Loads the symbols and the probes of the program to trace.
`lua`	Switches from command-line to Lua scripting environment. See below for a description of `Lua`.
`target [core]`	Sets or, when run with no parameters, shows the target core.

The command “`lua`” changes the input mode. The subsequent input is directed to a `Lua` interpreter on a per-line basis. The user can compose and execute singe-line `Lua` statements. Most CLI commands are available as `Lua` functions. To switch back to the CLI, call `cli()` in the `Lua` shell.

For example:

```
otrace> lua
otrace> help("system")
PP0:~CONSOLE-> 1. cli()
PP0:~CONSOLE-> Switch to cli mode.
PP0:~CONSOLE-> 2. exit()
PP0:~CONSOLE-> Exit otrace master.
PP0:~CONSOLE-> 3. help(["cmd|group"])
PP0:~CONSOLE-> Show help info for a cmd or cmd group (tracing, system, or misc).
PP0:~CONSOLE-> 4. ldsyms("se_app_bin")
PP0:~CONSOLE-> Load the symbols and the probes of the program to trace.
PP0:~CONSOLE-> 5. target([core])
PP0:~CONSOLE-> Set or, with no parameters, show the target core.
otrace> cli()
otrace> help system
PP0:~CONSOLE-> 1. exit
PP0:~CONSOLE-> Exit otrace master.
PP0:~CONSOLE-> 2. help [cmd|group]
PP0:~CONSOLE-> Show help info for a cmd or cmd group (tracing, system, or misc).
PP0:~CONSOLE-> 3. ldsyms se_app_bin
PP0:~CONSOLE-> Load the symbols and the probes of the program to trace.
PP0:~CONSOLE-> 4. lua
PP0:~CONSOLE-> Switch to lua mode.
PP0:~CONSOLE-> 5. target [core]
PP0:~CONSOLE-> Set or, with no parameters, show the target core.
```

Note: For more information on `Lua`, see R. Ierusalimschy. *Programming in Lua*. Lua.org, 2nd edition, 2006.

Many commands assume a target core, set by the command `target` with the core's number as the parameter. On the target, the core that runs the SE application under tracing cannot be the core on which the `otrace master` runs.

5.3.2 Tracing Commands

The Tracing commands include:

Table 5–2 otrace Tracing Commands

Option	Description
`fn_entry_clear func[ var1 var2 ...]`	Clears tracing at a function's entry on the target core.
`fn_entry_trace func[ var1 var2 ...]`	Sets tracing at a function's entry on the target core.
`fn_exit_clear func[ var1 var2 ...]`	Clears tracing at a function's exit on the target core.
`fn_exit_trace func[ var1 var2 ...]`	Sets tracing at a function's exit on the target core.
`shprobes [func]`	Shows probe info (for `func`) on the target core.
`shtrace max`	Displays a maximum of `max` log entries collected on the target core.
`trace [start|stop]`	Starts or stops tracing or shows if tracing is started on the target core.

Probe Operations

To list the probes, run the command `shprobes` as shown in the example below.

```
otrace> shprobes
PP0:~CONSOLE-> application_shutdown
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit off
PP0:~CONSOLE-> application_shutdown_simple_exec
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit off
PP0:~CONSOLE-> main
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit off
PP0:~CONSOLE-> application_main_loop
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit off
PP0:~CONSOLE-> application_init_simple_exec
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit off
PP0:~CONSOLE-> afn
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit off
PP0:~CONSOLE-> cycle_out_of_range
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit off
PP0:~CONSOLE-> hello_world
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit off
otrace> shprobes afn
PP0:~CONSOLE-> afn
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit off
```

This example shows all the probe locations and those of the function `afn` upon `otrace master`'s start. All the probes have the status `off` and are inactive.

To turn on a probe, for example, at the entry of `afn`:

```
otrace> fn_entry_trace afn
otrace> shprobes afn
PP0:~CONSOLE-> afn
PP0:~CONSOLE->    entry on
PP0:~CONSOLE->    exit off
```

When a probe is turned on, the timestamp is taken and recorded when the probe code is run.

Variables can be observed by inserting `tracers` for them in a probe. This is done using `fn_entry_trace` with the names of the variables. For example:

```
otrace> fn_entry_trace afn avar8 bvar8 cvar8
otrace> shprobes afn
PP0:~CONSOLE-> afn
PP0:~CONSOLE->     entry on
PP0:~CONSOLE->         cvar8
PP0:~CONSOLE->         bvar8
PP0:~CONSOLE->         avar8
PP0:~CONSOLE->     exit off
```

This traces three variables at the function `afn's` entry probe. When the probe code is executed, the values of the variables are read and recorded.

These tracers can be removed from the probe using `fn_entry_clear`. The probe can be disabled using the same command with only the function name.

For example:

```
otrace> fn_entry_clear afn cvar8
otrace> shprobes afn
PP0:~CONSOLE-> afn
PP0:~CONSOLE->     entry on
PP0:~CONSOLE->         bvar8
PP0:~CONSOLE->         avar8
PP0:~CONSOLE->     exit off
otrace> fn_entry_clear afn
otrace> shprobes afn
PP0:~CONSOLE-> afn
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit off
```

Tracing Operations

Once the interested events are specified, `trace start` actually kicks off tracing on the target and `shtrace max` can be executed to show the traces collected (See Understanding Output). When trace is executed without a parameter, it shows if tracing is enabled on the target core.

5.3.3 Miscellaneous Commands

The Miscellaneous commands include:

Table 5–3 otrace Miscellaneous Commands

Option	Description
`setvar var val`	Sets the value of a variable on the target core.
`shcop0 reg sel`	Shows the target core's `COP0` register value.
`shvar var`	Shows `var's` value on the target core.
`where`	Dumps the registers and show `call stack` on the target core.

The `where` command interrupts the target core to display its current context, including the content of the registers and the `call stack`.

For example:

```
otrace> where
r0 ($00): 0x0000000000000000     s0 ($16): 0x000000001000135c
at ($01): 0xfffffffffffbfffff     s1 ($17): 0x0000000010000000
v0 ($02): 0x0000000000000001     s2 ($18): 0x0001291de72d71f3
v1 ($03): 0x0001291de74c581b     s3 ($19): 0x0000000000000000
a0 ($04): 0x431bde82d7b634db     s4 ($20): 0x00000000100ff660
a1 ($05): 0x00000000100010c0     s5 ($21): 0xffffffffffffffff
a2 ($06): 0x000000001000135c     s6 ($22): 0x0000000000000000
a3 ($07): 0x000000000000000f     s7 ($23): 0x0000000000000000
t0 ($08): 0x0000000000000010     t8 ($24): 0x0000000000000000
t1 ($09): 0x0000000000000002     t9 ($25): 0x0000000000000017
t2 ($10): 0x0000000000000000     k0 ($26): 0x000000001001f0f8
t3 ($11): 0x0000000000000001     k1 ($27): 0xffffffff80001190
t4 ($12): 0x0000000000000000     gp ($28): 0x0000000012009180
t5 ($13): 0x0000000000000000     sp ($29): 0x00000000160fff30
t6 ($14): 0x8001180008000218     s8 ($30): 0x0000000000000000
t7 ($15): 0x00000000000d9101     ra ($31): 0x000000001000135c
     COP0_CAUSE: 0x0000000040008400
    COP0_STATUS: 0x00000000501000e1
  COP0_BADVADDR: 0x45a80f0290089683
       COP0_EPC: 0x0000000010001230
#0 0x1001049c
#1 0x1001dfe8
#2 0x1001f1c0
#3 0x10001230
#4 0x1000135c
#5 0x100006a8
#6 0x10000c48
```

Note: As of `otrace` v1.6, the target core does the printing.

5.4 Understanding Output

The `otrace stub` records runtime events and passes them to the `otrace master` as logs through a per-core ring buffer. As in a single producer-consumer model, the `stub` continues to append to the buffer until it is full, at which point it records a buffer full event. The `stub` continues to update the buffer full event until the `otrace master` consumes logs and makes room in the buffer.

As of otrace v1.6, each log entry obtained by the shtrace command contains the following items:

- **event type**—Identifies four types of events:
  - entry event generated upon entering a probed function.
  - exit event generated upon exiting a probed function.
  - variable event generated for a variable tracer in a probe.
  - buffer full event occurs when the buffer between the otrace master and the stub is full.
- **time stamp**—The cycle count upon the event.
- **type-specific content**—For function entry and exit events, contains the instrumented function's address and the address where it is called (for example, its call site). For buffer full events, contains the timestamp for the most recent attempt to record a log and the number of failed attempts so far due to buffer full.

Conceptually, a variable tracer resides in a probe. Therefore, the timestamp is only taken on the probe.

The output of an example trace is displayed below:

```
otrace> fn_entry_trace afn avar8 bvar8 cvar8
otrace> shprobes afn
PP0:~CONSOLE-> afn
PP0:~CONSOLE->     entry on
PP0:~CONSOLE->         cvar8
PP0:~CONSOLE->         bvar8
PP0:~CONSOLE->         avar8
PP0:~CONSOLE->     exit off
otrace> trace start
otrace> shtrace 10
fn <afn(0)> caller <application_main_loop(6740)> event <fn entry> [12858s:816432426(cycles)]
var addr <0x12005a88> value <0x0>
var addr <0x12005a78> value <0x0>
var addr <0x12005a68> value <0x0>
fn <afn(0)> caller <application_main_loop(6740)> event <fn entry> [12858s:822511218(cycles)]
var addr <0x12005a88> value <0x0>
var addr <0x12005a78> value <0x0>
var addr <0x12005a68> value <0x0>
fn <afn(0)> caller <application_main_loop(6740)> event <fn entry> [12858s:828588585(cycles)]
var addr <0x12005a88> value <0x0>
otrace> shtrace 1022
...
fn <afn(0)> caller <application_main_loop(6740)> event <fn entry> [12859s:186646953(cycles)]
var addr <0x12005a88> value <0x0>
event <buf full> [12862s:935655489(cycles)] starting [12859s:186647128(cycles)] #lost events=6926
fn <afn(0)> caller <application_main_loop(6740)> event <fn entry> [12862s:935655521(cycles)]
var addr <0x12005a88> value <0x0>
var addr <0x12005a78> value <0x0>
var addr <0x12005a68> value <0x0>
fn <afn(0)> caller <application_main_loop(6740)> event <fn entry> [12862s:938358057(cycles)]
var addr <0x12005a88> value <0x0>
var addr <0x12005a78> value <0x0>
var addr <0x12005a68> value <0x0>
fn <afn(0)> caller <application_main_loop(6740)> event <fn entry> [12862s:941060649(cycles)]
```

In these results, for each function entry event:

- afn(0) is the instrumented function (afn is the function name and (0) is the offset).
- application_main_loop(6740) is the caller.
- secs:cycles is the format for the timestamp.

Under each function entry event, the three traced variables' addresses and values are displayed. For the buffer full event, the two timestamps represent the last and first time where buffer full occurred; the #lost events value indicates 6926 events were lost during buffer full.

5.5 Example Traces

The following sections provide step-by-step examples of using otrace.

5.5.1 Example 1

1. **Define the following in `passthrough.c` and build the application.**

```
static uint8_t  foo8  __attribute__((used));
static uint16_t foo16 __attribute__((used));
static uint32_t foo32 __attribute__((used));
static uint64_t foo64 __attribute__((used));
static uint8_t  bar8  __attribute__((used));
static uint16_t bar16 __attribute__((used));
static uint32_t bar32 __attribute__((used));
static uint64_t bar64 __attribute__((used));
static void hello_world(void) __attribute__((used));        static void
hello_world(void)
     {
          cvmx_wait_usec(2000);
     }
```

2. **Launch otrace on core 0 and the application on other cores (on an OCTEON CN63xx).**

```
Octeon ep6300c# tftp 0 passthrough; bootoct 0 coremask=3e
Using octmgmt0 device
TFTP from server 13.14.15.1; our IP address is 13.14.15.2
Filename 'passthrough'.
Load address: 0x20000000
Loading: ##################################################
done
Bytes transferred = 4469099 (44316b hex)
Bootloader: Done loading app on coremask: 0x3e
Octeon ep6300c# tftp 0 otrace; bootoct 0 coremask=1
Using octmgmt0 device
TFTP from server 13.14.15.1; our IP address is 13.14.15.2
Filename 'otrace'.
Load address: 0x20000000
Loading:
 ##################################################
done
Bytes transferred = 5507607 (540a17 hex)
Bootloader: Done loading app on coremask: 0x1
eP0P:P1~:CO~NCOSNOSLOEL-E> -U>s iUsnign gde vdiecviec etr teer
  e
PP1:~CONSOLE-> Version: Cavium Inc. OCTEON SDK version 3.0.0, build 469
PP1:~CONSOLE-> Enabling CVMX_IPD_CTL_STATUS[NO_WPTR]
PP1:~CONSOLE-> Interface 0 has 4 ports (SGMII)
PP1:~CONSOLE-> Interface 3 has 4 ports (LOOP)
```

3. **Complete the following steps in otrace.**

   a. Check the probe status in `hello_world()`.

```
otrace> shprobes hello_world
PP0:~CONSOLE-> hello_world
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit
 off
```

b. Activate the entry probe and exit probe with variables.

```
otrace> fn_entry_trace hello_world
otrace> fn_exit_trace hello_world foo16
otrace> fn_exit_trace hello_world bar64
otrace> shprobes hello_world
PP0:~CONSOLE-> hello_world
PP0:~CONSOLE->     entry on
PP0:~CONSOLE->     exit on
PP0:~CONSOLE->         bar64
PP0:~CONSOLE->         foo16
```

No trace collected at this point as `hello_world()` is not called anywhere.

```
otrace> trace start
otrace> shtrace 10
```

c. Call `hello_world()` on the target core from `otrace master`.

```
otrace> crosscall hello_world
```

d. Review results from having probes executed once.

```
otrace> shtrace 10
fn <hello_world(0)> caller <otc_call(dc)> event <fn entry>
[326s:26011579(cycles)]
fn <hello_world(0)> caller <otc_call(dc)> event <fn exit>
[326s:28512046(cycles)]
var addr <0x12005a34> value <0x0>
var addr <0x12005a18> value <0x0>
```

e. Set the value of `bar64`.

```
otrace> setvar bar64 1234
otrace> shvar bar64
PP0:~CONSOLE-> 0x4d2
```

f. Call `hello_world()` again.

```
otrace> crosscall hello_world
otrace> shtrace 20
fn <hello_world(0)> caller <otc_call(dc)> event <fn entry>
[609s:40795486(cycles)]
fn <hello_world(0)> caller <otc_call(dc)> event <fn exit>
[609s:43295641(cycles)]
var addr <0x12005a34> value <0x0>
var addr <0x12005a18> value <0x4d2>
```

5.5.2 Example 2

1. **Build `passthrough` and `otrace-linux_64`.**

   Build `passthrough` (cvmx_64) with `-finstrument-functions`. This instruments virtually everything. Build the `otrace master` with `OCTEON_TARGET=linux_64`.

2. **Set up.**

```
cn52xx        <--------------------------------> cn63xx
(traffic-gen)     4 GBE ports back-to-back     (passthrough on 3 cores
                                                 and
                                                 Linux + otrace on core 0)
```

3. **Boot the target.**

```
tftp 0 passthrough; bootoct 0 coremask=0xe; tftp 0 vmlinux.64; bootoctlinux 0
coremask=1
```

4. **Start otrace after shipping `otrace-linux_64` and `passthrough` to Linux,**

```
~ # chmod 700 otrace-linux_64
~ # ./otrace-linux_64
CVMX_SHARED: 0x1202f0000-0x120320000
Active coremask = 0x1
```

5. **Import `passthrough`'s symbols into `otrace master`.**

```
otrace> ldsyms passthrough
```

6. **Review the probes of `cvmx_wqe_get_port()`.**

```
otrace> shprobes cvmx_wqe_get_port@0x10001ba0
cvmx_wqe_get_port@0x10001ba0
    entry off
    exit off
otrace> shprobes cvmx_wqe_get_port@0x10013cd8
cvmx_wqe_get_port@0x10013cd8
    entry off
    exit off
```

The `0x10001ba0` and `0x10013cd8` are addresses of two instances of the inline function `cvmx_wqe_get_port()`. One can obtain these using `nm`; for example:

```
% mipsisa64-octeon-elf-nm passthrough|grep cvmx_wqe_get_port
10001ba0 t cvmx_wqe_get_port
10013cd8 t cvmx_wqe_get_port
```

You can also obtain these using the otrace `shprobes` command; for example:

```
otrace> shprobes wqe_get_port
PP0:~CONSOLE-> cvmx_wqe_get_port@0x10013cd8
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit off
PP0:~CONSOLE-> cvmx_wqe_get_port@0x10001ba0
PP0:~CONSOLE->     entry off
PP0:~CONSOLE->     exit off
```

7. **Activate the probes at the entry and exit of both instances.**

```
otrace> fn_entry_trace cvmx_wqe_get_port@0x10001ba0
otrace> fn_entry_trace cvmx_wqe_get_port@0x10013cd8
otrace> fn_exit_trace cvmx_wqe_get_port@0x10001ba0
otrace> fn_exit_trace cvmx_wqe_get_port@0x10013cd8
otrace> shprobes cvmx_wqe_get_port@0x10013cd8
cvmx_wqe_get_port@0x10013cd8
    entry on
    exit on
otrace> shprobes cvmx_wqe_get_port@0x10001ba0
cvmx_wqe_get_port@0x10001ba0
    entry on
    exit on
```

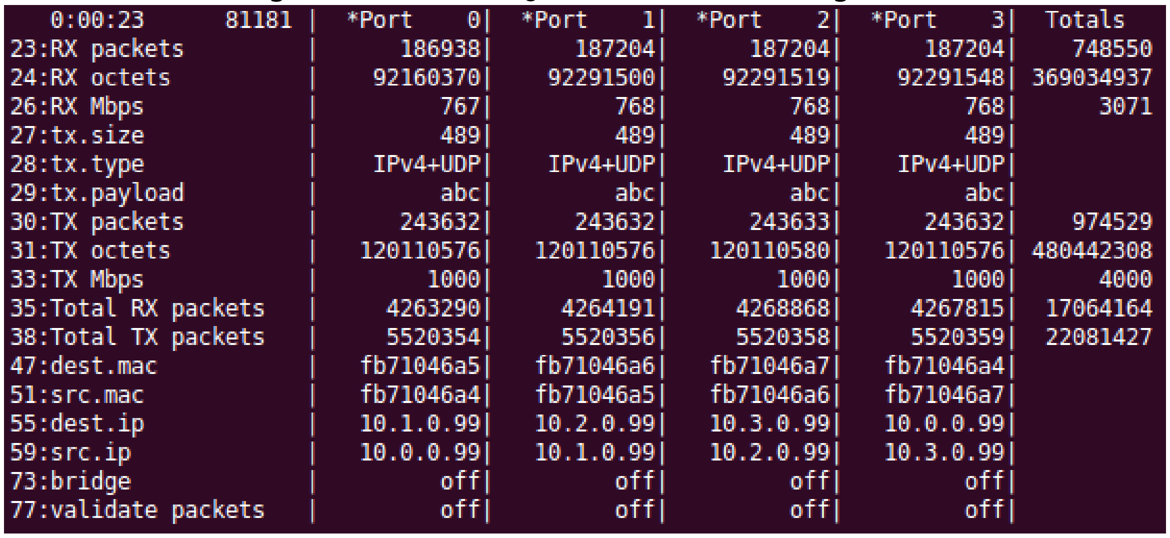
8. **Start tracing.**

```
otrace> trace start
otrace> shtrace 10
otrace>
```

Nothing returns immediately, so start pumping packets in `traffic-gen`:

```
otrace> shtrace 10
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> \
  event <fn entry> [521s:1020886065(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> \
  event <fn exit> [521s:1020886657(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> \
  event <fn entry> [521s:1020897379(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> \
  event <fn exit> [521s:1020897733(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> \
  event <fn entry> [521s:1020907514(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> \
  event <fn exit> [521s:1020907883(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> \
  event <fn entry> [521s:1020917533(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> \
  event <fn exit> [521s:1020917878(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> \
  event <fn entry> [521s:1020927528(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> \
  event <fn exit> [521s:1020927897(cycles)]
```

Figure 5–2 `traffic-gen` Screen After Starting Trace

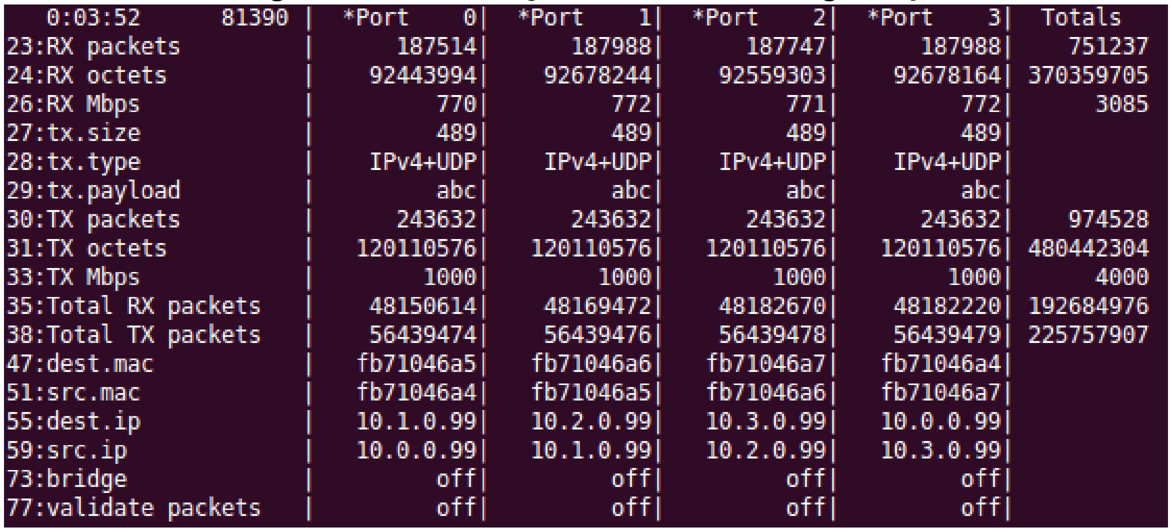


```
  0:00:23       81181 |  *Port     0|  *Port     1|  *Port     2|  *Port     3|  Totals
23:RX packets         |       186938|       187204|       187204|       187204|   748550
24:RX octets          |     92160370|     92291500|     92291519|     92291548| 369034937
26:RX Mbps            |          767|          768|          768|          768|     3071
27:tx.size            |          489|          489|          489|          489|
28:tx.type            |     IPv4+UDP|     IPv4+UDP|     IPv4+UDP|     IPv4+UDP|
29:tx.payload         |          abc|          abc|          abc|          abc|
30:TX packets         |       243632|       243632|       243633|       243632|   974529
31:TX octets          |    120110576|    120110576|    120110580|    120110576| 480442308
33:TX Mbps            |         1000|         1000|         1000|         1000|     4000
35:Total RX packets   |      4263290|      4264191|      4268868|      4267815|  17064164
38:Total TX packets   |      5520354|      5520356|      5520358|      5520359|  22081427
47:dest.mac           |     fb71046a5|     fb71046a6|     fb71046a7|     fb71046a4|
51:src.mac            |     fb71046a4|     fb71046a5|     fb71046a6|     fb71046a7|
55:dest.ip            |    10.1.0.99|    10.2.0.99|    10.3.0.99|    10.0.0.99|
59:src.ip             |    10.0.0.99|    10.1.0.99|    10.2.0.99|    10.3.0.99|
73:bridge             |          off|          off|          off|          off|
77:validate packets   |          off|          off|          off|          off|
```

9. **Trace only the entry.**

```
otrace> trace stop
otrace> fn_exit_clear cvmx_wqe_get_port@0x10001ba0
otrace> shprobes cvmx_wqe_get_port@0x10001ba0
cvmx_wqe_get_port@0x10001ba0
    entry on
    exit off
otrace> trace start
otrace> shtrace 10
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> event \
  <fn entry> [699s:856353998(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> event \
  <fn entry> [699s:856363993(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> event \
  <fn entry> [699s:856373962(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> event \
  <fn entry> [699s:856383922(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> event \
  <fn entry> [699s:856393880(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> event \
  <fn entry> [699s:856403863(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> event \
  <fn entry> [699s:856413820(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> event \
  <fn entry> [699s:856423778(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> event \
  <fn entry> [699s:856433736(cycles)]
fn <cvmx_wqe_get_port@0x10001ba0(0)> caller <application_main_loop(1dc)> event \
  <fn entry> [699s:856443718(cycles)]
```

Figure 5–3 `traffic-gen` Screen After Tracing `entry`

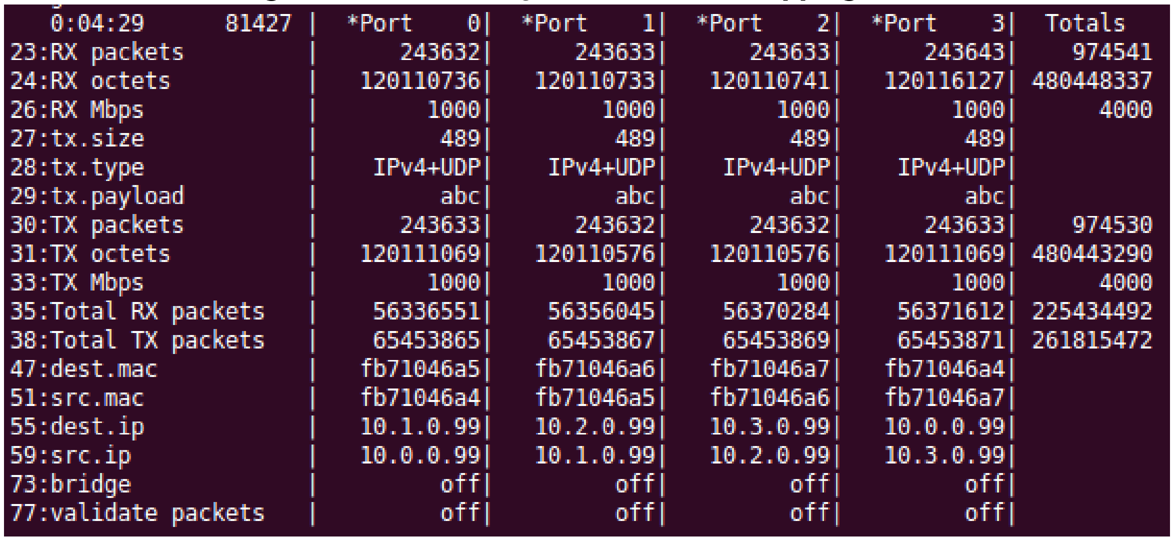


```
  0:03:52       81390 |  *Port     0|  *Port     1|  *Port     2|  *Port     3|   Totals
23:RX packets         |       187514|       187988|       187747|       187988|    751237
24:RX octets          |     92443994|     92678244|     92559303|     92678164| 370359705
26:RX Mbps            |          770|          772|          771|          772|      3085
27:tx.size            |          489|          489|          489|          489|
28:tx.type            |     IPv4+UDP|     IPv4+UDP|     IPv4+UDP|     IPv4+UDP|
29:tx.payload         |          abc|          abc|          abc|          abc|
30:TX packets         |       243632|       243632|       243632|       243632|    974528
31:TX octets          |    120110576|    120110576|    120110576|    120110576| 480442304
33:TX Mbps            |         1000|         1000|         1000|         1000|      4000
35:Total RX packets   |     48150614|     48169472|     48182670|     48182220| 192684976
38:Total TX packets   |     56439474|     56439476|     56439478|     56439479| 225757907
47:dest.mac           |    fb71046a5|    fb71046a6|    fb71046a7|    fb71046a4|
51:src.mac            |    fb71046a4|    fb71046a5|    fb71046a6|    fb71046a7|
55:dest.ip            |    10.1.0.99|    10.2.0.99|    10.3.0.99|    10.0.0.99|
59:src.ip             |    10.0.0.99|    10.1.0.99|    10.2.0.99|    10.3.0.99|
73:bridge             |          off|          off|          off|          off|
77:validate packets   |          off|          off|          off|          off|
```

10. **Stop the trace.**

```
otrace> trace stop
otrace>
```

Figure 5–4 `traffic-gen` Screen After Stopping Trace



```
   0:04:29      81427 |  *Port    0|  *Port    1|  *Port    2|  *Port    3|  Totals
23:RX packets         |      243632|      243633|      243633|      243643|    974541
24:RX octets          |   120110736|   120110733|   120110741|   120116127| 480448337
26:RX Mbps            |        1000|        1000|        1000|        1000|      4000
27:tx.size            |         489|         489|         489|         489|
28:tx.type            |    IPv4+UDP|    IPv4+UDP|    IPv4+UDP|    IPv4+UDP|
29:tx.payload         |         abc|         abc|         abc|         abc|
30:TX packets         |      243633|      243632|      243632|      243633|    974530
31:TX octets          |   120111069|   120110576|   120110576|   120111069| 480443290
33:TX Mbps            |        1000|        1000|        1000|        1000|      4000
35:Total RX packets   |    56336551|    56356045|    56370284|    56371612| 225434492
38:Total TX packets   |    65453865|    65453867|    65453869|    65453871| 261815472
47:dest.mac           |   fb71046a5|   fb71046a6|   fb71046a7|   fb71046a4|
51:src.mac            |   fb71046a4|   fb71046a5|   fb71046a6|   fb71046a7|
55:dest.ip            |   10.1.0.99|   10.2.0.99|   10.3.0.99|   10.0.0.99|
59:src.ip             |   10.0.0.99|   10.1.0.99|   10.2.0.99|   10.3.0.99|
73:bridge             |         off|         off|         off|         off|
77:validate packets   |         off|         off|         off|         off|
```

Chapter 6

The perf Tool

6.1 Introduction

Perf is a profiling tool for Linux 2.6 and higher-based systems based on the `perf_events` interface. The tool provides abstractions for hardware-specific capabilities to measure performance through a simple command-line interface.

This section provides specific information about the mapping between perf hardware events and OCTEON Performance Control Registers. For a complete description of how to use, refer to the tutorial on the perf wiki (https://perf.wiki.kernel.org/index.php/Tutorial).

6.2 Event Mapping

Perf maps hardware events to OCTEON Performance Control Registers as described in the following table:

Table 6–1 Events and Register Mapping

Perf Event	OCTEON Performance Control Register	Description (supported events from OCTEON HRM)
`cpu-cycles OR cycles`	[PERF_CNT_CLK] (reported in HZ)	Conditionally clocked cycles (as opposed to `count/cvm_count`, which counts even with no clocks)
`stalled-cycles-frontend OR idle-cycles-frontend`	N/A	
`stalled-cycles-backend OR idle-cycles-backend`	N/A	
`instructions`	[PERF_CNT_RET]	Instructions retired
`cache-references`	[PERF_CNT_LDS]	Number of load issues
`cache-misses`	[PERF_CNT_DMLDS]	Number of loads that were not prefetched and missed in the cache
`branch-instructions OR branches`	[PERF_CNT_BR]	Branches retired, excluding branch likelies
`branch-misses`	[PERF_CNT_BRMIS]	Branch mispredicts, excluding branch likelies
`bus-cycles`	[PERF_CNT_BADD]	Number of address bus cycles used (may need to set CvmCtl[DISCE] for accurate counts)
`ref-cycles`	N/A	

Perf Event	OCTEON Performance Control Register	Description (supported events from OCTEON HRM)
`L1-dcache-loads`	[PERF_CNT_LDS]	Number of load issues
`L1-dcache-load-misses`	[PERF_CNT_DMLDS]	Number of loads that were not prefetched and missed in the cache
`L1-dcache-stores`	[PERF_CNT_STS]	Number of store issues
`L1-dcache-store-misses`	N/A	
`L1-dcache-prefetches`	N/A	
`L1-dcache-prefetch-misses`	N/A	
`L1-icache-loads`	[PERF_CNT_CFETCH]	Icache committed fetches (demand+prefetch)
`L1-icache-load-misses`		
`L1-icache-prefetches`	[PERF_CNT_CPREF]	Icache committed prefetches
`L1-icache-prefetch-misses`	N/A	
`LLC-loads`	N/A	
`LLC-load-misses`	N/A	
`LLC-stores`	N/A	
`LLC-store-misses`	N/A	
`LLC-prefetches`	N/A	
`LLC-prefetch-misses`	N/A	
`dTLB-loads`	[PERF_CNT_DTLB]	Number of dstream TLB refill, invalid, or modified exceptions
`dTLB-load-misses`	[PERF_CNT_DTLB]	Number of dstream TLB refill, invalid, or modified exceptions
`dTLB-stores`	[PERF_CNT_DTLB]	Number of dstream TLB refill, invalid, or modified exceptions
`dTLB-store-misses`	[PERF_CNT_DTLB]	Number of dstream TLB refill, invalid, or modified exceptions
`dTLB-prefetches`	[PERF_CNT_DTLB]	Number of dstream TLB refill, invalid, or modified exceptions
`dTLB-prefetch-misses`	[PERF_CNT_DTLB]	Number of dstream TLB refill, invalid, or modified exceptions
`iTLB-loads`	[PERF_CNT_ITLB]	Number of dstream TLB refill, invalid, or modified exceptions
`iTLB-load-misses`	[PERF_CNT_ITLB]	Number of dstream TLB refill, invalid, or modified exceptions

Perf Event	OCTEON Performance Control Register	Description (supported events from OCTEON HRM)
`branch-loads`	[PERF_CNT_J]	Jumps retired; does not include J/JAL
`branch-load-misses`	[PERF_CNT_JMIS]	Jumps mispredicted; ddoes not include J/JAL
`node-loads`	N/A	
`node-load-misses`	N/A	
`node-stores`	N/A	
`node-store-misses`	N/A	
`node-prefetches`	N/A	
`node-prefetch-misses`	N/A	

6.3 Reading Additional Counters

Some OCTEON counters are not mapped to `perf`. These can be read from `/proc/octeon_perf`.

Important! `/proc/octeon_perf` and `perf` step over each other when reading counters; consequently, they should not be used simultaneously.

Chapter 7

The perfzilla Utility

7.1 Introduction

The perfzilla utility is a post-profiling graphical tool that analyzes the results from `oct-remote-profile` or `oct-linux-profile`. It derives meaningful information from the profile logs collected over a longer period of time using the profiling tools.

7.2 Features

The perfzilla utility includes the following features:

- Provides a Graphical User Interface (GUI).
- Displays the sorted lists of files and functions in descending order based on the Hit Count in the **File List** window.
- Displays the sample percentage and sample count against the file names in the **File List** window.
- Displays the source code of the corresponding file in the **Source Code** window.
- When a file is double clicked, displays the first hit count in the **Source Code** window; displays subsequent hits on every double click.
- Displays view of **Next Hit** and **Previous Hit** by selecting the corresponding action from **View** menu.
- Supports per core/set of cores sample views.
- Supports copy, paste, and search options.
- Supports mouse wheel option for all text widgets and scroll bars in the window.

7.3 Getting Started

The perfzilla utility requires the following two files as input:

- `profile.log`—Contains the captured output of `oct-pci-profile` or a `Viewzilla` trace.
- `ELF` file—Contains Dwarf2 debugging information.

To start perfzilla, run the following commands:

```
$ sudo oct-remote-profile > profile.log
$ perfzilla profile.log elf_file
```

This prompts perfzilla to complete the following steps:

- Load the `ELF` symbols.
- Load the line information of supplied `elf_file.`

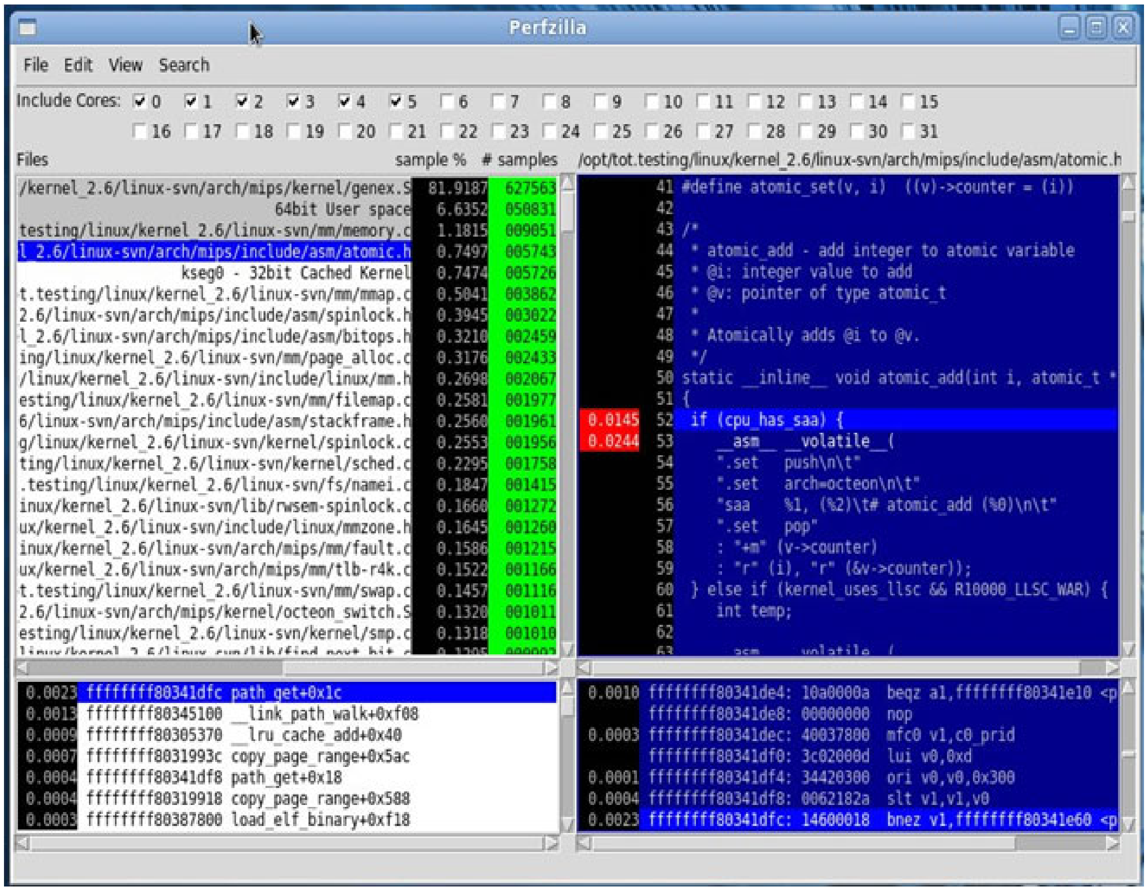
- Create a temporary file `perfzilla_objdump_elf_file.tmp` (the `objdump -d` of the `elf_file`).
- Read the profile information from the supplied `profile.log` file.

After these steps are complete, the perfzilla GUI is loaded.

7.4 The User Interface

The perfzilla Graphical User Interface is illustrated below:

Figure 7–1 perfzilla GUI



Include Cores: [✓ 0] [✓ 1] [✓ 2] [✓ 3] [✓ 4] [✓ 5] 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

```
/kernel_2.6/linux-svn/arch/mips/kernel/genex.S   81.9187  627563
                               64bit User space   6.6352  050831
testing/linux/kernel_2.6/linux-svn/mm/memory.c    1.1815  009051
l_2.6/linux-svn/arch/mips/include/asm/atomic.h    0.7497  005743
                      kseg0 - 32bit Cached Kernel 0.7474  005726
t.testing/linux/kernel_2.6/linux-svn/mm/mmap.c    0.5041  003862
2.6/linux-svn/arch/mips/include/asm/spinlock.h    0.3945  003022
l_2.6/linux-svn/arch/mips/include/asm/bitops.h    0.3210  002459
ing/linux/kernel_2.6/linux-svn/mm/page_alloc.c    0.3176  002433
/linux/kernel_2.6/linux-svn/include/linux/mm.h    0.2698  002067
esting/linux/kernel_2.6/linux-svn/mm/filemap.c    0.2581  001977
6/linux-svn/arch/mips/include/asm/stackframe.h    0.2560  001961
g/linux/kernel_2.6/linux-svn/kernel/spinlock.c    0.2553  001956
ting/linux/kernel_2.6/linux-svn/kernel/sched.c    0.2295  001758
.testing/linux/kernel_2.6/linux-svn/fs/namei.c    0.1847  001415
inux/kernel_2.6/linux-svn/lib/rwsem-spinlock.c    0.1660  001272
ux/kernel_2.6/linux-svn/include/linux/mmzone.h    0.1645  001260
inux/kernel_2.6/linux-svn/arch/mips/mm/fault.c    0.1586  001215
ux/kernel_2.6/linux-svn/arch/mips/mm/tlb-r4k.c    0.1522  001166
t.testing/linux/kernel_2.6/linux-svn/mm/swap.c    0.1457  001116
2.6/linux-svn/arch/mips/kernel/octeon_switch.S    0.1320  001011
esting/linux/kernel_2.6/linux-svn/kernel/smp.c    0.1318  001010
```

/opt/tot.testing/linux/kernel_2.6/linux-svn/arch/mips/include/asm/atomic.h

```
        #define atomic_set(v, i)  ((v)->counter = (i))

        /*
         * atomic_add - add integer to atomic variable
         * @i: integer value to add
         * @v: pointer of type atomic_t
         *
         * Atomically adds @i to @v.
         */
        static __inline__ void atomic_add(int i, atomic_t *
        {
0.0145   if (cpu_has_saa) {
0.0244      __asm__ __volatile__(
            ".set   push\n\t"
            ".set   arch=octeon\n\t"
            "saa    %1, (%2)\t# atomic_add (%0)\n\t"
            ".set   pop"
            : "+m" (v->counter)
            : "r" (i), "r" (&v->counter));
         } else if (kernel_uses_llsc && R10000_LLSC_WAR) {
            int temp;
```

```
0.0023 ffffffff80341dfc path_get+0x1c
0.0013 ffffffff80345100 __link_path_walk+0xf08
0.0009 ffffffff80305370 __lru_cache_add+0x40
0.0007 ffffffff8031993c copy_page_range+0x5ac
0.0004 ffffffff80341df8 path_get+0x18
0.0004 ffffffff80319918 copy_page_range+0x588
0.0003 ffffffff80387800 load_elf_binary+0xf18
```

```
0.0010 ffffffff80341de4: 10a0000a  beqz a1,ffffffff80341e10 <p
       ffffffff80341de8: 00000000  nop
0.0003 ffffffff80341dec: 40037800  mfc0 v1,c0_prid
       ffffffff80341df0: 3c02000d  lui v0,0xd
0.0001 ffffffff80341df4: 34420300  ori v0,v0,0x300
0.0004 ffffffff80341df8: 0062182a  slt v1,v1,v0
0.0023 ffffffff80341dfc: 14600018  bnez v1,ffffffff80341e60 <p
```

The perfzilla GUI includes the following elements:

- Main Menu
- Include Cores Pane
- perfzilla Windows

7.4.1 Main Menu

The **Main** menu includes the following items:

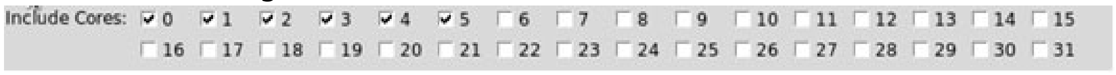
Menu	Option
File	**New Window**—Opens a new instance of `perfzilla` using the same `ELF` and `profile.log` files as the current instance.
	Save Source—Saves the source code file with profiling information.

Menu	Option
Edit	**Copy**—Copies the text from any of the four windows. This text can be pasted into a text editor or into an email.
View	**Previous Hit**—Moves the cursor to the previous sample information in source file.
	Next Hit—Moves the cursor to the next sample information in the source code.
	Function Profile Info—Displays the detailed profiling information.
Search	**Find**—Searches (greps) for a matching string in the source code.
	Find next—Searches (greps) for the next matching string in the source code.

7.4.2 Include Cores Pane

The **Include Cores** pane displays the maximum number of cores available on OCTEON. In the following figure, the maximum number of cores is 32:

Figure 7–2 Include Cores Pane



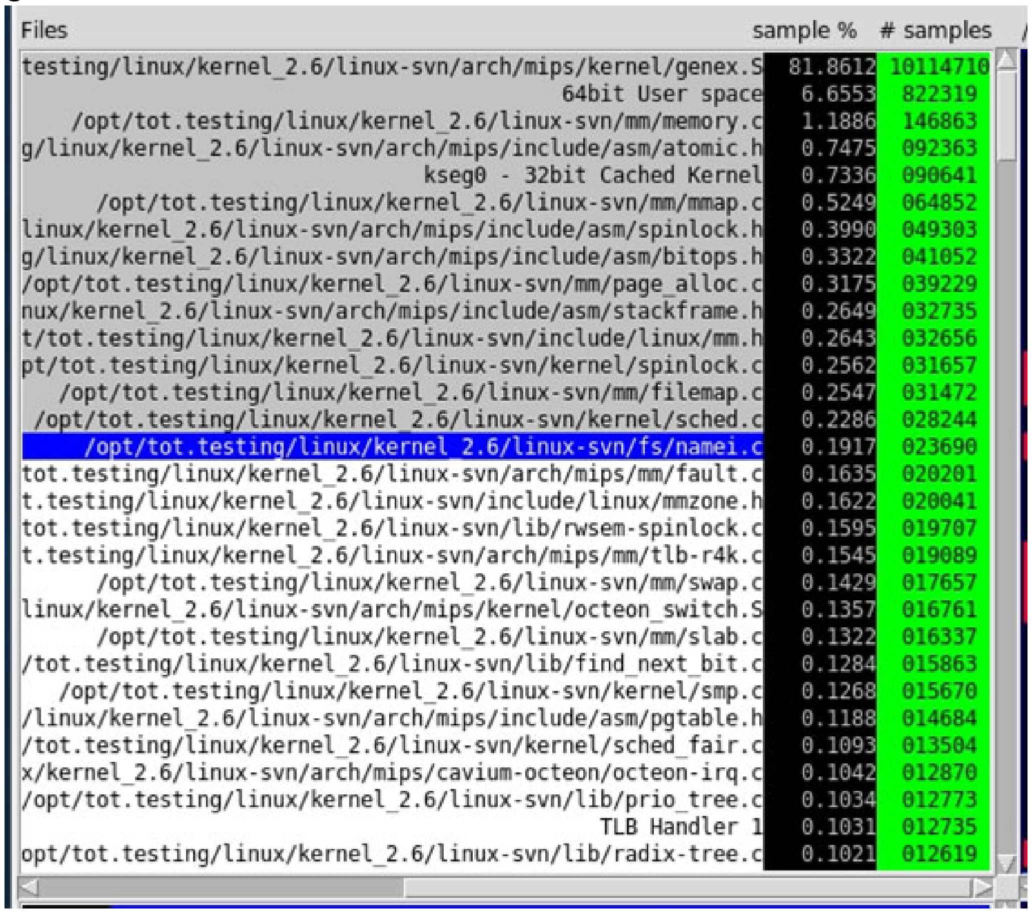
The selected check boxes represent the number of cores that are currently being profiled. The result of total samples or sample% varies depending on the number of cores being selected or deselected in this section. This is useful to control the sampling information for a single core or subset of cores.

7.4.3 perfzilla Windows

The perfzilla GUI includes the following windows:

- **File List** window (upper left)
- **Source Code** window (upper right)
- **Function List** window (lower left)
- **ASM** window (lower right)

Figure 7–3 File List Window



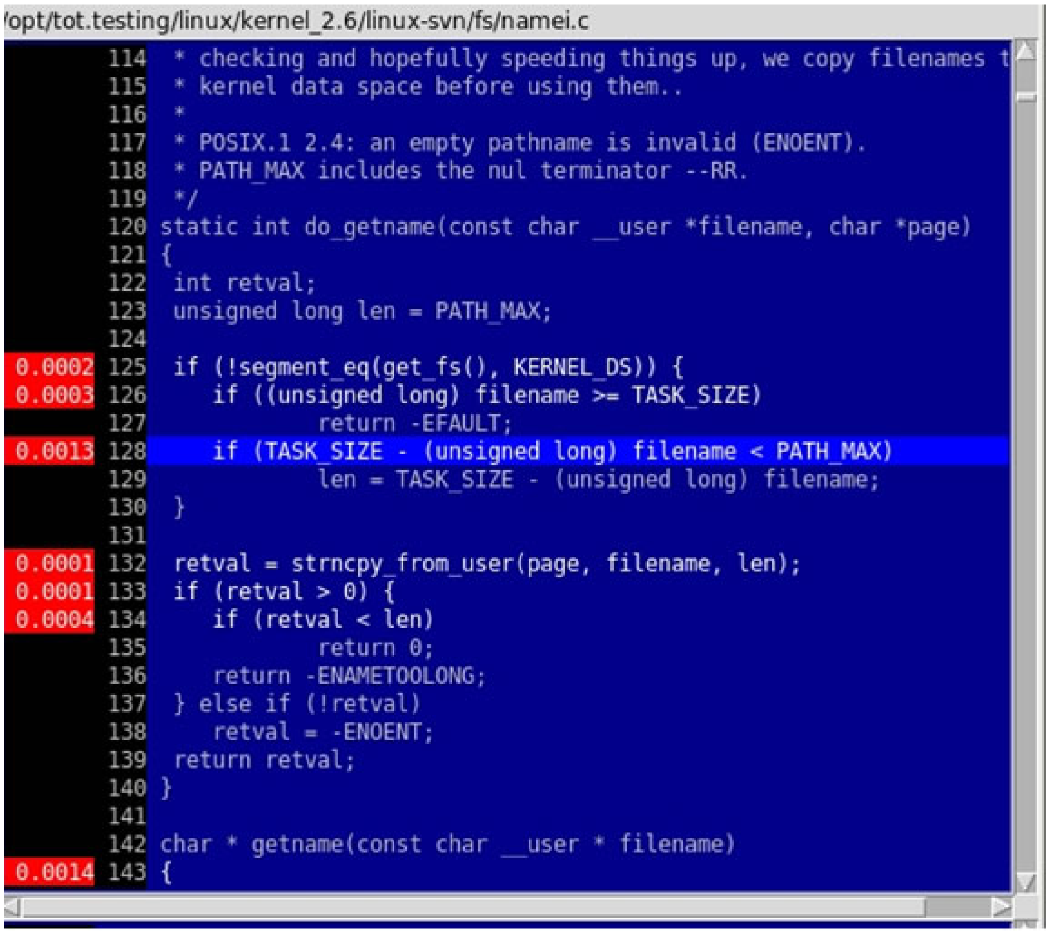
Files	sample %	# samples
testing/linux/kernel_2.6/linux-svn/arch/mips/kernel/genex.S	81.8612	10114710
64bit User space	6.6553	822319
/opt/tot.testing/linux/kernel_2.6/linux-svn/mm/memory.c	1.1886	146863
g/linux/kernel_2.6/linux-svn/arch/mips/include/asm/atomic.h	0.7475	092363
kseg0 - 32bit Cached Kernel	0.7336	090641
/opt/tot.testing/linux/kernel_2.6/linux-svn/mm/mmap.c	0.5249	064852
linux/kernel_2.6/linux-svn/arch/mips/include/asm/spinlock.h	0.3990	049303
g/linux/kernel_2.6/linux-svn/arch/mips/include/asm/bitops.h	0.3322	041052
/opt/tot.testing/linux/kernel_2.6/linux-svn/mm/page_alloc.c	0.3175	039229
nux/kernel_2.6/linux-svn/arch/mips/include/asm/stackframe.h	0.2649	032735
t/tot.testing/linux/kernel_2.6/linux-svn/include/linux/mm.h	0.2643	032656
pt/tot.testing/linux/kernel_2.6/linux-svn/kernel/spinlock.c	0.2562	031657
/opt/tot.testing/linux/kernel_2.6/linux-svn/mm/filemap.c	0.2547	031472
/opt/tot.testing/linux/kernel_2.6/linux-svn/kernel/sched.c	0.2286	028244
/opt/tot.testing/linux/kernel_2.6/linux-svn/fs/namei.c	0.1917	023690
tot.testing/linux/kernel_2.6/linux-svn/arch/mips/mm/fault.c	0.1635	020201
t.testing/linux/kernel_2.6/linux-svn/include/linux/mmzone.h	0.1622	020041
tot.testing/linux/kernel_2.6/linux-svn/lib/rwsem-spinlock.c	0.1595	019707
t.testing/linux/kernel_2.6/linux-svn/arch/mips/mm/tlb-r4k.c	0.1545	019089
/opt/tot.testing/linux/kernel_2.6/linux-svn/mm/swap.c	0.1429	017657
linux/kernel_2.6/linux-svn/arch/mips/kernel/octeon_switch.S	0.1357	016761
/opt/tot.testing/linux/kernel_2.6/linux-svn/mm/slab.c	0.1322	016337
/tot.testing/linux/kernel_2.6/linux-svn/lib/find_next_bit.c	0.1284	015863
/opt/tot.testing/linux/kernel_2.6/linux-svn/kernel/smp.c	0.1268	015670
/linux/kernel_2.6/linux-svn/arch/mips/include/asm/pgtable.h	0.1188	014684
/tot.testing/linux/kernel_2.6/linux-svn/kernel/sched_fair.c	0.1093	013504
x/kernel_2.6/linux-svn/arch/mips/cavium-octeon/octeon-irq.c	0.1042	012870
/opt/tot.testing/linux/kernel_2.6/linux-svn/lib/prio_tree.c	0.1034	012773
TLB Handler 1	0.1031	012735
opt/tot.testing/linux/kernel_2.6/linux-svn/lib/radix-tree.c	0.1021	012619

The **File List** window displays:

- A list of source files whose functions had at least one sample registered during profiling.
- The sample% and sum total of all samples collected against a particular file.
- The absolute file name path.
- Horizontal and vertical scroll bars to make it easy to navigate through the contents.

When you select a file in the **File List** window, the source code for the file displays in the **Source Code** window. The **Sample%** (in red) displays opposite a particular instruction or C source code line.

Figure 7–4 Source Code Window

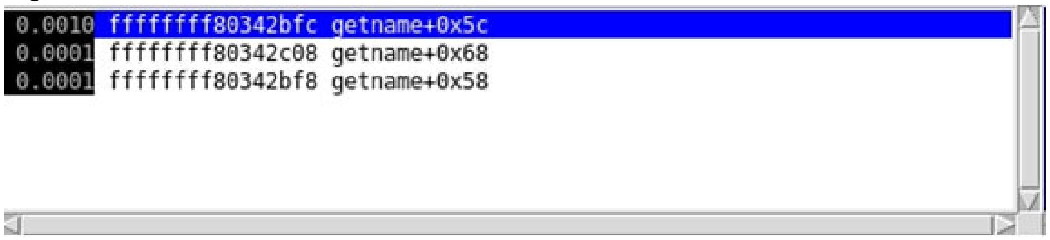


```
/opt/tot.testing/linux/kernel_2.6/linux-svn/fs/namei.c
        * checking and hopefully speeding things up, we copy filenames t
        * kernel data space before using them..
        *
        * POSIX.1 2.4: an empty pathname is invalid (ENOENT).
        * PATH_MAX includes the nul terminator --RR.
        */
       static int do_getname(const char __user *filename, char *page)
       {
        int retval;
        unsigned long len = PATH_MAX;

0.0002  if (!segment_eq(get_fs(), KERNEL_DS)) {
0.0003     if ((unsigned long) filename >= TASK_SIZE)
                   return -EFAULT;
0.0013     if (TASK_SIZE - (unsigned long) filename < PATH_MAX)
                   len = TASK_SIZE - (unsigned long) filename;
        }

0.0001  retval = strncpy_from_user(page, filename, len);
0.0001  if (retval > 0) {
0.0004     if (retval < len)
                   return 0;
           return -ENAMETOOLONG;
        } else if (!retval)
           retval = -ENOENT;
        return retval;
       }

       char * getname(const char __user * filename)
0.0014 {
```

You can take the following actions from the **Source Code** window:

View samples in the file:	• Double-click the file name in the **File List** window • Select **View** > **Next hit** (ALT+Down in **Source Code** window) • Select **View** > **Previous hit** (ALT+Up in **Source Code** window)
Search for a pattern in the source file:	Select **Search** > **Find** or **Search** > **Find Next** (F5 in Source Code window). The search pattern is highlighted in yellow.
Copy and paste source code into a text editor or email:	Select **Edit** > **Copy** (CTRL+C), right-click, and select **Paste** from the context menu.

Figure 7–5 Function List Window



```
0.0010 ffffffff80342bfc getname+0x5c
0.0001 ffffffff80342c08 getname+0x68
0.0001 ffffffff80342bf8 getname+0x58
```

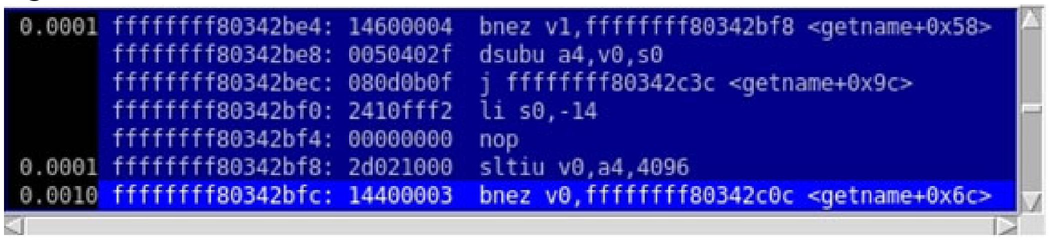
The **Function List** window displays:

- The PC value and function name of the C source code line selected in the **Source Code** window.

  Note: This value may not be 100% accurate.

- The sample% against the PC values and routine name.

Figure 7–6 ASM Window



```
0.0001 ffffffff80342be4: 14600004  bnez v1,ffffffff80342bf8 <getname+0x58>
       ffffffff80342be8: 0050402f  dsubu a4,v0,s0
       ffffffff80342bec: 080d0b0f  j ffffffff80342c3c <getname+0x9c>
       ffffffff80342bf0: 2410fff2  li s0,-14
       ffffffff80342bf4: 00000000  nop
0.0001 ffffffff80342bf8: 2d021000  sltiu v0,a4,4096
0.0010 ffffffff80342bfc: 14400003  bnez v0,ffffffff80342c0c <getname+0x6c>
```

The **ASM** window includes the following information:

- Displays the disassembly of the instruction or C code line selected in the **Source Code** window.
- Updates the window with the sample% count and disassembly of the C instruction.

The display may span multiple lines depending on the C code. The instructions may not be continuous in some cases.

Note: The sample% and line number shown in the C Source code are approximate, and should not be considered 100% accurate, as the Complier optimizations will re-arrange instructions to avoid bubbles or stalls. Also note that if an instruction stalls, the sample% or count is calculated up to the line following the stalled instruction, instead of the instruction that actually stalled.

Chapter 8

SE-Dump and the oct-remote-coredump Tool

8.1 Introduction

SE-Dump and the oct-remote-coredump tool together enable the creation of a core dump file when an SE application crashes. This core file can then be analyzed using standard GDB commands.

SE-Dump is a new mechanism through which the exception handler creates an ELF core header in memory (reserved while initializing). Host/Linux side tools are used to fetch the memory location and dump the memory contents into an ELF core file.

An ELF core file is an ELF object file with a current snapshot of the memory contained in the file. Depending on the memory consumed by the application, this file can grow enormously. With an ELF core file and GDB, the user can debug the problem with source code. The available debug view is greatly improved with stack trace availability, and to view the values of global/local variables at each level of function call sequence.

The oct-remote-coredump tool is a helper tool written to coordinate with the core dumping facility built into Simple Executive. Once the SE application runs into an exception, the default behavior is to create a core dump of the program. The user can then dump the entire snapshot of the program's memory to analyze offline to discover where the error occurred.

8.2 Features

The oct-remote-coredump tool includes the following features:

- Capable of dumping multi-core application.
- Creates a fully GDB- capable core dump binary for analysis.
- Capable of using all protocols supported by `remote-lib` (`PCI`, `JTAG`, and so on).

8.3 How the Tool Works

When an exception occurs, the exception handler installed during initialization is called. In the exception handler, `cvmx_coredump_start_dump()` is called.This calculates the address at which the ELF header has to be created. The ELF header address depends on the core number. The macros defined in `cvmx-coredump.h` and `cvmx-coredump-elf.h` specify that the memory location is used to dump the ELF header along with related information.

1. The SE application's exception handler calculates the offset for saving registers.
2. `SE-Dump` starts to build an ELF program header, for which it needs the TLB entries. Through the TLB entries, it determines code/data and stack information for populating the ELF header.
3. Once the TLB and registers are in place, dumping is started.
4. The ELF header is created.

5. A program header is created with one `PT_NOTE` section and the value of (the number of TLBs + the number of memory allocations).

Once the program headers are in place, `oct-remote-coredump` runs to complete the process. The oct-remote-coredump tool reads the ELF header along with program headers. First, the ELF header is written, then the program headers, and then the `PT_NOTE` section. Finally, for each entry in the program header, it copies the target's memory to file.

8.4 Command-line Option

The supported command-line option for oct-remote-coredump is:

```
oct-remote-coredump <coremask>
```

where `<coremask>` is the mask used to run the program. If no mask was provided (using `boot-oct`), `0x1` should be used. This argument is mandatory.

8.5 Getting Started

The sources for the oct-remote-coredump utility are located in the `${OCTEON_ROOT}/host/coredump-util` directory. SE-Dump is enabled by default in Simple Executive applications, so no action is required to enable it.

The following steps describe how to use SE-Dump and oct-remote-coredump to generate core dump files for analysis with GDB:

1. **Build the SE application.**

   Build your own SE application, or use the sample program provided at `<OCTEON_ROOT>/examples/coredump` for trying out the tool.

   To build the sample program (`coredump`), run the following commands:

   ```
   $ cd <OCTEON_ROOT>/examples/coredump
   $ make
   ```

2. **Load and run the application.**

   Transfer the resulting binary from step 2 to the target (for details on loading the application, see the applicable sections of the *OCTEON® Programmer's Guide, The Fundamentals* (Curtis, 2009). At the U-boot prompt, run the command:

   ```
   Octeon-ebb5200# bootoct $(loadaddr) coremask=0x3 endbootargs
   ```

   The sample program is designed to crash after calculating some `checksums` of `SHA1`, `MD5`, and `Kasumi`. Once the program crashes, you can run the `oct-remote-coredump` tool.

3. **Dump the corefile.**

   Run the following command (from either the host or on a target running Linux):

   ```
   $ oct-remote-coredump 0x3.
   ```

   This creates sequentially named files —`CVMX_DUMP.CORE_0` and `CVMX_DUMP.CORE_1`—for `core0` and `core1` (which is specified in the command line by `coremask 0x3`).

4. **Run `gdb` to analyze the crash global/local variables.**

   Because each core has different data/stack segments allocated while running the application, it is impossible to encode all the information related to the application in the ELF file format. Consequently, separate core files are generated for each core. You must repeat this step for each core.

An example GDB session is shown below.

Table 8–1 gdb session

```
$ mips64-octeon-linux-gnu-gdb
<<snip>>
(gdb) symbol-file coredump-crypto
Reading symbols from examples/coredump/coredump-crypto...done.
(gdb) core-file CVMX_DUMP.CORE_0
[New process 1]
#0 0x0000000010000d94 in crash () at coredump-crypto.c:745
745                         *p = 0x1234;
(gdb) backtrace
#0 0x0000000010000d94 in crash () at coredump-crypto.c:745
#1 0x0000000010001294 in test_snow3g () at coredump-crypto.c:796
#2 test_kasumi () at coredump-crypto.c:829
#3 test_aes () at coredump-crypto.c:861
#4 0x0000000000000000 in ?? ()
(gdb) info registers
  Id  Target Id    Frame
* 1   process 1    0x0000000010000d94 in crash () at coredump-crypto.c:745

                  zero               at               v0               v1
R0    0000000000000000 ffffffffffbfffff 0000000000001234 000000000000000a
                    a0               a1               a2               a3
R4    ffffffffffffffff 000000000000000a 0000000000000019 0000000000000019
                    a4               a5               a6               a7
R8    ffffffffff7f0000 0a0a0a0a0a0a0a0a 000000000000002a 0000000000000000
                    t0               t1               t2               t3
R12   0000000000000012 0000000000000000 0000000000000000 0000000000000000
                    s0               s1               s2               s3
R16   0000000012016150 0000000000000001 00000000101c9328 ffffffffc0784fb0
                    s4               s5               s6               s7
R20   0000000000000028 0000000000000000 0000000000000000 0000000000000000
                    t8               t9               k0               k1
R24   0000000000000000 0000000010000238 0000000010034748 ffffffff80001090
                    gp               sp               s8               ra
R28   00000000120091d0 00000000160fff98 0000000016100000 0000000010000784
                    sr               lo               hi              bad
      00000000501004e3 0000000000000000 0000000000000000 0000000000000000
                 cause               pc
      000000004000800c 0000000010000d94
                   fsr              fir
      gdb-script0:5: Error in sourced command file:
can't read register 160 (fsr)
(gdb) print plaintext
$1 = 0x101c9328 "Original plaintext message"
(gdb)
```

Chapter 9

The viewzilla Tool

9.1 Introduction

The OCTEON SDK includes the performance profiling tool viewzilla, which runs on the OCTEON hardware simulator. The tool collects profiling data and provides a visual interface for interpreting that data, showing the application programmer where the code can be optimized for OCTEON.

Important! viewzilla only runs on the hardware simulator; it does not run on actual hardware.

9.2 Features

The viewzilla tool includes the following features:

- Provides a visual representation of each processor's pipeline.
- Handles detailed load for each processor's crypto engine (`-modes=cryptovz` for the simulator).
- Manages Level 1 and Level 2 cache operations.
- Controls coherent bus utilization.
- Utilizes low latency memory bandwidth.
- Displays C Source.
- Performs instruction tracing per processor or for all processors.
- Enables fast searching.
- Provides loop visualization with instruction stepping.
- Facilitates quick scale view from high-level overview down to individual instructions.

9.3 Getting Started

The viewzilla tool requires the following two files as input:

- `octeon.vz`—the viewzilla file to load. The file is generated by running `oct-sim` with the `-vz` option. The option `-modes=cryptovz` can also be used to detail the crypto units.
- `elf`—Optional Mips ELF files from which to load debugging information.

To start viewzilla, run the following command:

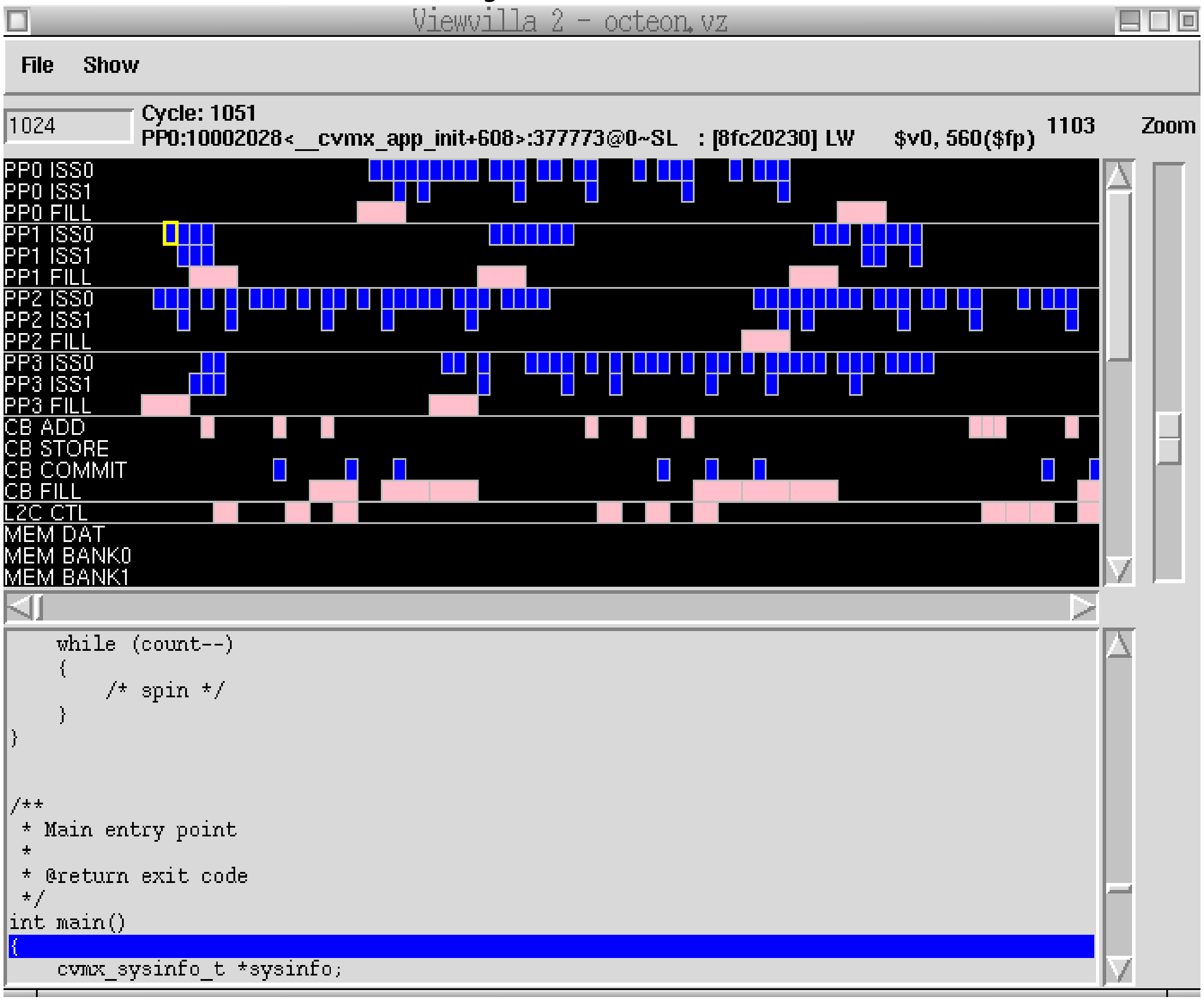
```
$ viewzilla octeon.vz [elf] [elf] ...
```

To use viewzilla, build applications with DWARF debugging information (`-g` compiler option).

9.4 The User Interface

The viewzilla Graphical User Interface is illustrated below:

Figure 9–1 viewzilla GUI



```
Cycle: 1051
PP0:10002028<__cvmx_app_init+608>:377773@0~SL  : [8fc20230] LW     $v0, 560($fp)
```

```
    while (count--)
    {
        /* spin */
    }
}


/**
 * Main entry point
 *
 * @return exit code
 */
int main()
{
    cvmx_sysinfo_t *sysinfo;
```

The viewzilla GUI includes the following elements:

- Main Menu
- Windows and Navigation Tools

9.4.1 Main Menu

The **Main** menu includes the following items:

Table 9–1 viewzilla Main Menu

Menu	Option
File	**Find** — Used to find arbitrary strings in the `viewzilla` file. Searching starts at the current cycle and continues forward through time. Searches are case sensitive. Short cut keys: **Control-F**
	Find Next — Continues the previous search. Short cut keys: **Control-G, n**.
	Quit — Exits the `viewzilla` program.
Show	**Source**—Uses the DWARF debugging information from the `ELF` files to show the original program source.
	Processor Instructions—Displays the instruction trace annotated by source for the currently selected processor.
	All Processor Instructions —Displays the instruction traces annotated by source for all the running processors.
	Highlight Next—Moves the highlight to the next active cycle for the selected hardware unit. Short cut keys: **->**, **k**.
	Highlight Previous—Moves the highlight to the previous active cycle for the selected hardware unit. Short cut keys: **<-**, **j**.
	Hardware—Controls whether each hardware unit is displayed.

9.4.2 Windows and Navigation Tools

Other windows and navigation tools on the viewzilla interface are described in the following table:

Table 9–2 viewzilla Windows and Navigation Tools

Menu	Option
Start cycle input field	The first clock cycle displayed at the left edge of the cycle view window. This value may be modified by directly typing a value, scrolling the horizontal scrollbar, or by using one of the searching operations.
Selected box description	Displays a short description of what is occurring whenever the mouse moves over an active cycle.
Ending cycle	The last cycle visible in the cycle view window. It is calculated based on the current zoom level and window width.
Cycle view window	Displays each hardware unit (vertical axis) and its state at each cycle (horizontal axis). Moving the mouse over boxes shows a short description. Click a box to select it and any boxes associated with it; available source/instructions are displayed. Step to the next/previous selection using the left and right arrow keys.
Vertical scrollbar for the cycle view window	If the viewzilla window is too small, not all hardware units can be display in the cycle view window. Use this scrollbar to control which ones are available.
Zoom level slider	Controls how much information is displayed. Slide up to show fewer cycles, but more detail. Slide down to show more cycles, but less detail.

Menu	Option
Horizontal scrollbar for the cycle view window	Quickly move to any cycle in the viewzilla trace. The left and right arrows move one cycle at a time. Click the page back/forward arrows to move 3/4s of the view area at a time.
Source / Instruction display window	Depending on the selection in the **Show** pulldown menu, displays either the source file for the current instruction or an instruction trace with source annotation. **Note:** This is only available when an instruction is selected.
Vertical scrollbar for the source window	Scrolls through the loaded source file or instruction trace.

9.5 Understanding Output

When you click a processor instruction in the viewzilla interface to select it, text detailing the state of the processor at execution time displays. The following examples explain how to decode the text.

Example 1:

```
PP0:20001254<main+116>:2400@0~MisP : [24a50001] ADDIU $a1, $a1, 1
[$a1=2]
```

- `PP0`—The core to which the information applies; in this case, core 0
- `20001254<main+116>`—The instruction virtual address (hex + label)
- `2400@0`—Cycle 2400, issued on pipe 0
- `~MisP`—viewzilla code indicating that the instruction was delayed due to a branch mispredict
- `24a50001`—Hex of the instruction
- `ADDIU $a1, $a1, 1`—Instruction mnemonic

Example 2:

```
PP0:20001264<main+132>:7302@0~DWB : [dfbf0000] LD $ra, 0($sp)
[$ra=20000264(7FFFFFD0,7FFFFFD0)]
```

- `~DWB`—viewzilla code indicating that this load missed and matched a pending write buffer
- `(7FFFFFD0,7FFFFFD0)`—(virtual address,physical address) in hex
- `$ra=20000264`—0x20000264 was loaded from physical address 0x7FFFFFD0

Example 3:

```
PP0:200004EC<flush_print+ 36>:7383@0~SL : [ffb00000] SD $s0,
0($sp) [(7FFFFFD0,7FFFFFD0)=0]
```

- `~SL`—viewzilla code indicating that the instruction was delayed because it had to issue on pipe 0
- `(7FFFFFD0,7FFFFFD0)`— (virtual address,physical address) in hex
- `=0`—The value 0 was written to physical address 0x7FFFFFD0

9.5.1 Delay/Stall Codes

The following table provides a list of codes that viewzilla can return due to delays or stalls:

Table 9–3 viewzilla: List of Delay/Stall Codes

Stall	Reasons
`~ITLB`	Instruction TLB miss
`~BR`	Prior branch instruction
`~JR`	Prior jump instruction
`~MisP`	Prior branch mispredicted
`~MTCOP0`	Prior MT COP0 instruction
`~IF`	Instruction fetch
`~SL`	Needs pipe 0, but pipe 1 is available
`~BRF`	Ignore this
`~RAW`	Input register values not available
`~MisJR`	Mispredicted jump
`~WAI`	WAIT instruction
`~MUL`	Multiplier busy
`~CR`	Crypto busy
`~LLM`	LLM busy
`~D`	Dcache miss that does not hit a write buffer entry
`~DWB`	Dcache miss that hits a write buffer entry
`~SY`	SYNC/SYNCS/SYNCIOBDMA instruction

9.6 The vz-cut Utility

The `vz-cut` utility provides an easy way to copy a section of a viewzilla file into a smaller file. The original file is not modified.

To start `vz-cut`, run the following command:

```
$ vz-cut --start start_cycle --end end_cycle [--filename
input_filename] [--outfilename output_filename] [--help]
```

where

- `--start`—The cycle on which to start copying; defaults to `0`
- `--end`—The cycle on which to end copying; defaults to the end of the file
- `--filename`—Input filename; defaults to `octeon.vz`
- `--outfilename`—Output filename; defaults to '`octeon-cut.vz`
- `--help`—Displays help message.

Note: Only one of `start` or `end` is required because `start` defaults to `0` and `end` defaults to the end of the file if the other value is not provided.

9.7 Tips for Running viewzilla

The following list includes tips for running viewzilla effectively:

- Instruction tracing is already contained in the viewzilla file. To suppress tracing, run `oct-sim` with the `-quiet` option.
- Most of the time you are only interested in application performance. Use the option `-wait=main` to skip all analysis before the main entry point of the program. Without this option, it can be difficult to wade through all the output for the bootloader and C library startup.
- The simulator runs much more slowly with viewzilla output turned on. Use the `-wait` option to start output only in the area in which you are interested.
- viewzilla output files can get large very quickly. Make sure you have adequate free disk space. As an example, booting 16 cores and running for 10 million cycles generates a 14 GB output file.
- Compile applications to be traced with debugging symbols (`-g`).
- viewzilla uses the full path names present in the DWARF debugging information. Make sure source code can be found at these locations. If the source view is unable to find a file, it displays the path where it tried to locate the source.
- Due to the large amount of data, searching through a viewzilla trace can be slow. Avoid long searches by scrolling to the approximate cycle where your search result is expected to be found.

Appendix A

Controlling SE Applications from Linux

This Appendix provides instructions for using the oct-app-ctl utility, included in the SDK for controlling Simple Executive (SE) applications from Linux. Potential uses for controlling standalone SE applications from Linux include:

- Packaging the entire application, including control and data plane, in a single embedded root file system, or enabling a "live" upgrade of the data plane portion without rebooting the control plane.
- Dynamic load balancing where the number of CPU cores can be switched between the data plane SE application and Linux to match changing load patterns dynamically. In such cases, the dynamic load-control supervisor (not provided) would employ the Linux CPU hot-plugging mechanism to make CPU cores available from Linux and added to SE applications and vice versa.

Note: This utility replaces the bootoct and oct-shutdown-app utilities included in previous releases of the SDK.

A.1 Using the oct-app-ctl Utility

The oct-app-ctl utility supports five operations, described in the following sections:

- Starting an SE Application
- Displaying Application Status
- Shutting Down an Application
- Adding Cores to an Application
- Deleting Cores from an Application

For each operation, the exact action can be specified either as an argument to `oct-app-ctl` or by using a link to the utility executable identifying the operation as follows:

```
oct-app-ctl <operation>
```

or

```
oct-app-<operation>
```

A.1.1 Starting an SE Application

To start a SE application, run one of the following commands:

```
oct-app-ctl boot <options> <app_file_name> [<app_arguments>]
oct-app-boot <options> <app_file_name> [<app_arguments>]
```

The supported command-line options for oct-app-ctl when booting an application are listed in the following table.

Table A–1 Command-line Options for Booting an Application

Option	Description
`-verbose=<num>`	Increase verbosity level of the program up to the value of 3. Default is 0.
`-numcores=<N>`	The number of CPU cores on which to start the application. Default is 1.
`-mask=<hex-number>`	The core mask of CPUs on which to start the application.
`-heapsize=<bytes>`	Set the size of the heap memory allocated per CPU core. Default is 3 MB.
`-stacksize=<bytes>`	Set the size of the stack memory allocated per CPU core. Default is 1 MB.

Note the following when starting applications:

- This method can be used to start any SE application, whether or not the application supports hot-plugging (the remaining operations require the support of hot-plugging).
- The core mask argument (`mask=<hex-number>`) only supports up to 64 cores. Consequently, if the core mask argument is specified and includes cores that are not available, the unavailable cores are replaced with the same number of available cores to satisfy the core count in the requested mask, if possible.

A.1.2 Displaying Application Status

To display the status of applications, run one of the following commands:

```
oct-app-ctl info
oct-app-info
```

The command returns the following information:

- A list of which SE applications are currently running on which cores.

  Note: The cores currently being used by Linux are not specified. In order to control and display the CPU cores running Linux, refer to the Linux CPU hot-plugging documentation.

- Active core mask—Applications that are capable of supporting hot-plugging, which is required for shutting down applications and adding/removing cores.
- An index entry for every instance of an application. This information is useful if there are multiple applications with the same name (because they were started from the same executable file) and you wish to perform an operation on a specific instance of the application.
- The core mask of currently unused CPU cores that are available for starting new applications or adding into existing applications.

A.1.3 Shutting Down an Application

To shut down an application on all CPU cores on which it is running, run one of the following commands:

```
oct-app-ctl shutdown <options> {-index=<app-index> | <app-name>}
oct-app-shutdown <options> {-index=<app-index> | <app-name>}
```

The following option is available (it is not required):

Table A–2 Command-line Options for Shutting Down Applications

Option	Description
`-verbose=<num>`	Increase verbosity level of the program up to the value of 3. Default is 0.

Note the following when shutting down an application:

- The application being stopped must support hot-plugging and must have call-back routines installed so that it can be notified of requests for relinquishing or adding resources. See the example program located at `sdk/examples/hotplug_app` for additional information.
- The application affected must be identified either by its name, which is identical to the name of the executable object file that was used to start the application, or by means of the application instance index number.
- The executable file does not need to be present in the current (or any other) directory for this operation to succeed, as it is not accessed for anything other than the `boot` operation.

A.1.4 Adding Cores to an Application

To add cores to an application, run one of the following commands:

```
oct-app-ctl add <options> {-index= <app-index> | <app-name>}
oct-app-add <options> {-index= <app-index> | <app-name>}
```

The following table lists the options available when adding cores.

Table A–3 Command-line Options for Adding Cores

Option	Description
`-verbose=<num>`	Increase verbosity level of the program up to the value of 3. Default is 0.
`-numcores=<N>`	The number of CPU cores on which to start the application. Default is 1.
`-mask=<hex-number>`	The core mask of CPUs on which the application should be started.

Note the following when adding cores:

- The application must support hot-plugging and must have call-back routines installed so that it can be notified of requests for relinquishing or adding resources. See the example program located at `sdk/examples/hotplug_app` for additional information.
- The core mask argument (`mask=<hex-number>`) only supports up to 64 cores; it is recommended that the `-numcores` argument be used instead.
- If the core mask argument is specified and includes cores that are not presently available, the unavailable cores are replaced with the same number of available cores to satisfy the core count in the requested mask, if possible.

A.1.5 Deleting Cores from an Application

To delete cores from an application, run one of the following commands:

```
oct-app-ctl del <options> {-index= <app-index> | <app-name>}
oct-app-del <options> {-index= <app-index> | <app-name>}
```

The following table lists the options available when deleting cores.

Table A–4 Command-line Options for Deleting Cores

Option	Description
`-verbose=<num>`	Increase verbosity level of the program up to the value of 3. Default is 0.
`-numcores=<N>`	The number of CPU cores to relinquish. Default is 1.
`-mask=<hex-number>`	The coremask of CPUs that should be stopped.

Note the following when deleting cores:

- The application must support hot-plugging and must have call-back routines installed so that it can be notified of requests for relinquishing or adding resources. See the example program located at `sdk/examples/hotplug_app` for additional information.
- This operation will refuse to delete the initial core running the application, which is the lowest-numbered core on which the application was initially started. All other cores can be deleted, including cores that are numerically lower than the initial core but were added using the `add` operation after the application was started.
- If a core mask is specified in the command line that includes cores not running an instance of the application, or if the mask specifies the initial core, the actual mask is modified to stop the same number of cores as requested in the mask, if possible.

A.2 Limitations

The oct-app-ctl utility has the following limitations:

- The utility presently does not support bare-metal applications that have been started from the bootloader. It cannot add cores, remove cores, or shut such applications down. If deploying Linux, it is recommended that the bootloader only be used to start the kernel, with any further control over bare-metal SE applications performed from Linux.
- The utility has only been tested with big-endian executables and hosts. It does not currently support little-endian versions of either.

Appendix B

Examples

B.1 `oct-profile`

B.1.1 Invoking `oct-profile` for `traffic-gen`

`oct-profile` invoked for `traffic-gen` (over PCIe).

Table B–1 oct-profile: Invoking

```
Oct-profile /tftpboot/traffic-gen
core#:   0      1      2      3      4      5
      2356      0      0      0      0      0   main
      2356      0      0      0      0      0   statistics_gatherer
         0   3184   3184   3184   3184   3184   packet_transmitter
         0    162    166    138    157    161   cvmx_get_cycle
         0    162    166    138    157    161   cvmx_clock_get_count
         0     14     17     21     22     20   cvmx_send_single
         0     14     17     21     22     20   cvmx_write64_uint64
         0      0      0      0      0    170   cvmx_scratch_read64
         0    137    116    127    121    146   cvmx_scratch_write64
       489      0      0      0      0      0   periodic_update
        20      0      0      0      0      0   cvmx_get_cycle
        20      0      0      0      0      0   cvmx_clock_get_count
       339      0      0      0      0      0   uart_printf
```

B.1.2 Profiling `traffic-gen` over PCI

Profile `traffic-gen` running over PCI; Display PC along with sample hits associated with routines.

Table B–2 oct-profile: Profiling traffic-gen over PCI

```
oct-profile –address /tftpboot/traffic-gen
core#: 0      1     2     3     4     5
       1      0     0     0     0     0     __sfvwrite_r
       1      0     0     0     0     0     0x00000000101418a0
       2      0     0     0     0     0     vfprintf_r
       1      0     0     0     0     0     0x000000001013bfe0
       1      0     0     0     0     0     0x000000001013c610
    2719      0     0     0     0     0     main
    2719      0     0     0     0     0     statistics_gatherer
      17      0     0     0     0     0     0x0000000010028730
      13      0     0     0     0     0     0x000000001002873c
      18      0     0     0     0     0     0x0000000010028740
      24      0     0     0     0     0     0x0000000010028744
      81      0     0     0     0     0     0x0000000010028748
      23      0     0     0     0     0     0x0000000010028750
     860      0     0     0     0     0     0x0000000010028754
       5      0     0     0     0     0     0x000000001002875c
     119      0     0     0     0     0     0x0000000010028764
     146      0     0     0     0     0     0x000000001002876c
      24      0     0     0     0     0     0x000000001002878c
      32      0     0     0     0     0     0x0000000010028790
       2      0     0     0     0     0     0x0000000010028798
       4      0     0     0     0     0     0x000000001002879c
       7      0     0     0     0     0     0x00000000100287a0
    1344      0     0     0     0     0     0x00000000100287a4
       0   3636  3636  3636  3636  3636     packet_transmitter
       0    100   117   120   129   220     0x0000000010009410
       0     90    88    99    96   176     0x0000000010009418
       0    268   259   269   289   494     0x000000001000941c
       0     95   105   109    82   165     0x000000001000961c
       0    378   297   278   256   492     0x0000000010009624
       0     83   117   106    90   192     0x0000000010009628
       0    293   287   299   258   554     0x000000001000962c
       0     87    95   105    94   162     0x0000000010009634
```

```
        0     287   294   276   305   515     0x000000001000971c
        0      15    22    19    23    26     0x0000000010009728
        0     263   263   300   266     0     0x0000000010009760
        0     297   301   267   281     0     0x0000000010009764
        0      87    75    87    89     0     0x000000001000976c
        0     264   254   262   294     0     0x0000000010009770
        0      95    81    77    88     0     0x0000000010009e94
        0     105   107    97   101     0     0x0000000010009e9c
        0     185   189   182   184     0     0x0000000010009ea0
        0     244   251   224   243     0     0x0000000010009ea4
        0     191   181   187   193   170     cvmx_get_cycle
        0     191   181   187   193   170     cvmx_clock_get_count
        0      96    94    94   105   170     0x0000000010009618
        0      95    87    93    88     0     0x0000000010009e90
        0      81    97   113   136   108     cvmx_pow_work_request_async_nocheck
        0      46    53    57    85    70     0x0000000010009738
        0      20    24    32    26    15     0x000000001000973c
        0      15    20    24    25    23     cvmx_send_single
        0      15    20    24    25    23     cvmx_write64_uint64
        0      15    20    24    25    23     0x0000000010009740
        0       0     0     0     0   184     cvmx_scratch_read64
        0       0     0     0     0   184     0x0000000010009718
        0     128   156   160   139   178     cvmx_scratch_write64
        0     128   156   160   139   178     0x0000000010009724
      573       0     0     0     0     0     periodic_update
       16       0     0     0     0     0     0x000000001001aaf0
        8       0     0     0     0     0     0x000000001001aaf4
        9       0     0     0     0     0     0x000000001001aaf8
       14       0     0     0     0     0     0x000000001001aafc
       19       0     0     0     0     0     0x000000001001ab00
       13       0     0     0     0     0     0x000000001001ab04
        9       0     0     0     0     0     0x000000001001ab08
       21       0     0     0     0     0     0x000000001001ad28
       35       0     0     0     0     0     0x000000001001ad2c
       28       0     0     0     0     0     0x000000001001ad30
        1       0     0     0     0     0     0x0000000010023a04
       34       0     0     0     0     0     cvmx_get_cycle
       34       0     0     0     0     0     cvmx_clock_get_count
       34       0     0     0     0     0     0x000000001001ab20
      341       0     0     0     0     0     uart_printf
        5       0     0     0     0     0     0x0000000010004f60
        1       0     0     0     0     0     0x0000000010004f9c
        1       0     0     0     0     0     0x0000000010004fb8
      170       0     0     0     0     0     0x0000000010004fd0
```

B.2 `oct-remote-profile`

B.2.1 Profiling `traffic-gen`

Profiling `traffic-gen` application running on OCTEON connected over PCIe.

Table B–3 oct-remote-profile: Profiling traffic-gen

```
$oct-remote-profile
              core#:     0     1     2     3     4     5
0x0000000010004f60:     4     0     0     0     0     0
0x0000000010004fd0:   170     0     0     0     0     0
0x0000000010004fd4:    87     0     0     0     0     0
0x0000000010004fd8:    74     0     0     0     0     0
0x0000000010005008:     4     0     0     0     0     0
0x0000000010005010:     2     0     0     0     0     0
0x0000000010009410:     0    99    84   104   128   194
0x0000000010009418:     0   101    88    99    83   168
0x000000001000941c:     0   222   231   230   234   456
0x0000000010009618:     0    65    84    86    95    59
0x000000001000961c:     0    69    80    99    99   158
0x0000000010009624:     0   334   277   245   265   469
0x0000000010009628:     0    82   111   112    90   140
0x000000001000962c:     0   242   253   262   245   507
0x0000000010009634:     0    85    82    94    90   159
0x0000000010009718:     0     0     0     0     0   176
0x000000001000971c:     0   236   256   242   256   461
0x0000000010009724:     0   150   142   136   124   114
0x0000000010009728:     0    27    14    21    11    18
0x0000000010009738:     0    47    54    54    60    65
0x000000001000973c:     0    13    18    17    12    17
0x0000000010009740:     0    18    13    20    15    17
                                             .
                                 .
                                 .
0x0000000010028764:   146     0     0     0     0     0
0x000000001002876c:   173     0     0     0     0     0
0x000000001002878c:    35     0     0     0     0     0
0x0000000010028790:    41     0     0     0     0     0
0x0000000010028798:     3     0     0     0     0     0
0x000000001002879c:     2     0     0     0     0     0
0x00000000100287a0:     2     0     0     0     0     0
0x00000000100287a4:  1168     0     0     0     0     0
0x000000001003dea8:     2     0     0     0     0     0
0x000000001013bfe0:     2     0     0     0     0     0
```

B.2.2 Hardware Statistics Profiling `traffic-gen`

Print only `perf` counters, OCTEON hardware statistics when Profiling `traffic-gen` over PCIe.

Table B–4 oct-remote-profile: Hardware Statistics

```
$ oct-top -s
Core counters:
brmis 63407[0], 28648414[1], 28648708[2], 28649755[3], 28649762[4], 28682185[5],
total (excluding 0) brmis 143278824,
sync 0[0], 0[1], 0[2], 0[3], 0[4], 0[5],
total (excluding 0) sync 0,

L2 statistics for TAD 0:
bus_xmc(addr) count:                 645
bus_xmd(store) count:                546
bus_rsc(commit) count:               645
bus_rsd(fill) count:                 644
bus_ioc(IO req) count:               644
bus_ior(IO req) count:               644
hit count:                        590258
miss count:                            0
lfb-wait-vab bus utilization:         0%
lfb-wait-lfb bus utilization:         0%

DRAM Statistics:
DRAM ops count:                142357388
dclk count:                5530240452959
utilization:                       0.00%

FPA Pool statistics:
FPA Pool 0 (Packet Buffers):     816 free
FPA Pool 1 (WQE buffers):        490 free
FPA Pool 2 (PKO cmd buffers):      0 free, PKO output queues 128

POW/SSO Qos Input Queue Statistics
Queue 0: WQE count  0, Buffers allocated  0
Queue 1: WQE count  0, Buffers allocated  0
Queue 2: WQE count  0, Buffers allocated  0
Queue 3: WQE count  0, Buffers allocated  0
Queue 4: WQE count  0, Buffers allocated  0
Queue 5: WQE count  0, Buffers allocated  0
Queue 6: WQE count  0, Buffers allocated  0
Queue 7: WQE count  0, Buffers allocated  0
Total WQEs in all QOS levels: 0 (in-unit & in-memory)
Total Free Buffers in all QOS levels: 243
```

B.2.3 Profiling `traffic-gen` Over Subset of Cores

Profile `traffic-gen` running over PCIe on subset of cores.

Table B–5 oct-remote-profile: Profiling traffic-gen over Subset of Cores

```
$ Oct-remote-profile --core_mask=0x23
            core#:     0     1     5
0x0000000010004f60:     1     0     0
0x0000000010004f74:     1     0     0
0x0000000010004fd0:   129     0     0
0x0000000010004fd4:    81     0     0
0x0000000010004fd8:    74     0     0
0x0000000010004fe0:     1     0     0
0x0000000010005008:     1     0     0
0x0000000010005010:     1     0     0
0x0000000010009410:     0    95   193
0x0000000010009418:     0    79   177
0x000000001000941c:     0   234   482
0x0000000010009618:     0    67   156
0x000000001000961c:     0    74   166
0x0000000010009624:     0   350   496
0x0000000010009628:     0    87   144
0x000000001000962c:     0   252   449
0x0000000010009634:     0    69   146
0x0000000010009718:     0     0   149
0x000000001000971c:     0   246   445
0x0000000010009724:     0   116   120
0x0000000010009728:     0    16    12
0x0000000010009738:     0    46    49
0x000000001000973c:     0    19    17
0x0000000010009764:     0   250     0
0x000000001000976c:     0    71     0
0x0000000010009770:     0   237     0
0x0000000010009e90:     0    80     0
                     .
          .
          .
0x00000000100287a0:     3     0     0
0x00000000100287a4:  1197     0     0
0x000000001003deac:     1     0     0
0x0000000010138040:     1     0     0
```

B.2.4 Available Events

List of available events.**`oct-remote-profile`: Available events:**

```
$ oct-remote-profile -l
clk                 issue               ret                 nissue
sissue              dissue              ifi                 br
brmis               j                   jmis                replay
iuna                trap                uuload              uustore
uload               ustore              ec                  mc
cc                  csrc                cfetch              cpref
ica                 ii                  ip                  cimiss
wbuf                wdat                wbufld              wbuffl
wbuftr              badd                baddl2              bfill
ddids               idids               didna               lds
lmlds               iolds               dmlds               sts
lmsts               iosts               iobdma              dtlb
dtlbad              itlb                sync                synciob
syncw               eretmis             likmis              hazard-trap
```

B.2.5 Event-based Profiling for `traffic-gen`

Start event-based profiler for `traffic-gen` (running on target over PCIe) for Cache miss event and sample rate of 100000.

Table B–6 oct-remote-profile: Event-based Profiling for traffic-gen

```
$ oct-remote-profile –events=cimiss:100000
             core#:     0     1     2     3     4     5
0x0000000010004f60:     3     0     0     0     0     0
0x0000000010004f74:     1     0     0     0     0     0
0x0000000010004fd0:   124     0     0     0     0     0
0x0000000010004fd4:    48     0     0     0     0     0
0x0000000010004fe0:     7     0     0     0     0     0
0x0000000010005008:     2     0     0     0     0     0
0x000000001000500c:     1     0     0     0     0     0
0x0000000010009410:     0    35    34    25    28   160
0x0000000010009418:     0    27    30    33    30    41
0x000000001000941c:     0    58    60    61    60   126
0x0000000010009618:     0    28    29    36    27   172
0x000000001000961c:     0    59    67    62    61   153
0x0000000010009624:     0   126   110   140   118   145
0x0000000010009628:     0    31    31    30    25   153
0x000000001000962c:     0    69    58    56    77   184
0x0000000010009634:     0     0     0     0     0   183
0x0000000010009718:     0     0     0     0     0     3
0x000000001000971c:     0    94   130    96    83   205
0x0000000010009724:     0    48    78    61    67   114
0x0000000010009728:     0    28    28    26    25    27
0x0000000010009738:     0    26    21    31    25    36
0x000000001000973c:     0    27    34    28    26    31
0x0000000010009760:     0   275   283   276   315     0
0x0000000010009764:     0   183   205   179   174     0
0x0000000010009770:     0   199   179   191   218     0
0x0000000010009e90:     0   105    96   105   102     0
0x0000000010009e94:     0   111    80   103   105     0
0x0000000010009e9c:     0    93    97    96    91     0
0x0000000010009ea0:     0    86    88    81    81     0
0x0000000010009ea4:     0   113    83   104    83     0
0x0000000010028754:  1186     0     0     0     0     0
0x000000001002876c:     1     0     0     0     0     0
0x00000000100287a0:     1     0     0     0     0     0
0x00000000100287a4:  1630     0     0     0     0     0
0x000000001013bfe0:     1     0     0     0     0     0
0x000000001013c5e0:     1     0     0     0     0     0
^C
Event          :  cimiss
Sampling Rate  :  1000000
Total samples  :  17705
              Core#:     0     1     2     3     4     5
            Samples:  4433  2681  2680  2680  2680  2551
```

B.3 `oct-top`

B.3.1 Frequent Routines

List the frequently called routines (in `traffic-gen`) in increasing order of number of sample hits.

Table B–7 oct-top Frequent Routines

```
$ oct-top /tftpboot/traffic-gen
Function Name:                   Total hits
_vfprintf_r:                              1
cvmx_sysinfo_get:                         1
fdt_next_tag:                             2
x_scratch_read64:                         6
cvmx_get_cycle:                          17
cvmx_clock_get_count:                    17
cvmx_send_single:                       123
cvmx_write64_uint64:                    123
fastpath_receive:                       284
periodic_update:                        396
uart_printf:                            411
cvmx_scratch_write64:                   758
cvmx_get_cycle:                         846
cvmx_clock_get_count::                  846
main:                                  2893
statistics_gatherer:                   2893
packet_transmitter:                   18375
```

B.3.2 Top Information on a Core

Generate top information of an application running on a specific core.

Table B–8 oct-top: Top Core Information

```
$ oct-top /tftpboot/traffic-gen --cpu=0
Function Name:                      Total hits
cvmx_scratch_read64:                     1
cvmx_send_single:                       22
cvmx_write64_uint64:                    22
fastpath_receive:                       59
cvmx_get_cycle:                        184
cvmx_clock_get_count::                 184
cvmx_scratch_write64:                  189
packet_transmitter:                   3571
```

B.3.3 PC Values

List the PC values in ascending order.

Table B–9 oct-top: PC Values in Ascending Order

```
$ oct-top
0x0000000010004f8c:            1
0x00000000100078e0:            1
0x0000000010009a8c:            1
0x0000000010009fb4:            1
0x00000000100287e0:            1
0x0000000010140408:            1
0x0000000010005020:            2
0x0000000010005024:            2
0x000000001000774c:            2
0x00000000100077a8:            2
0x0000000010007918:            2
0x00000000100098b0:            2
0x00000000100099ac:            2
0x0000000010009a44:            2
0x0000000010009ca8:            2
0x0000000010009f64:            2
0x000000001000a02c:            2
0x00000000100287c4:            2
0x0000000010004f78:            3
0x0000000010007748:            3
0x00000000100077b8:            3
0x00000000100078f0:            3
0x000000001000790c:            3
0x0000000010007910:            3
                   .
                   .
0x000000001002876c:          804
0x000000001000963c:         1015
0x0000000010009ebc:         1052
0x0000000010009778:         1168
0x0000000010009434:         1170
0x000000001000977c:         1175
0x0000000010009734:         1204
0x0000000010009644:         1216
0x00000000100287bc:         1420
```

B.3.4 Frequently Called Routines and Source Code Information

List the frequently called routines and source code information associated with routine.

Table B–10 Frequently-called Routines

```
oct-top /tftpboot/traffic-gen --line
Function Name:                                                Total hits
fdt_check_node_offset:                                             1
/opt/tot.testing/executive/libfdt/fdt.c:134:                       1
cvmx_sysinfo_get:                                                  1
/opt/tot.testing/executive/cvmx-srio.c:405:                        1
fdt_offset_ptr:                                                    1
/opt/tot.testing/executive/libfdt/fdt.c:81:                        1
cvmx_scratch_read64:                                               5
/opt/tot.testing/target/include/cvmx-scratch.h:13:                 5
cvmx_get_cycle:                                                   16
cvmx_clock_get_count:                                             16
/opt/tot.testing/target/include/cvmx-clock.h:112:                 16
cvmx_send_single:                                                110
cvmx_write64_uint64:                                             110
/opt/tot.testing/target/include/cvmx-access-native.h:476:        110
fastpath_receive:                                                281
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:5864:         67
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:5866:         38
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:5931:         37
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:5877:         30
/opt/tot.testing/target/include/cvmx-wqe.h:582:                   25
/opt/tot.testing/target/include/cvmx-wqe.h:585:                   25
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:5924:         20
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:5869:         18
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:5883:         18
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:5684:          3
periodic_update:                                                 380
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:5320:        150
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:5341:        150
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:5322:         64
                                                                   .
                                                                   .
                                                                   .
packet_transmitter:                                            18259
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:6033:       4066
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:5989:       2361
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:6070:       2178
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:6029:       1553
/opt/tot.testing/examples/traffic-gen/traffic-gen.c:6026:       1531
```

B.3.5 Routines and Hexadecimal PC Values

List the routines and the Hexadecimal PC values associated with routines.

Table B–11 oct-top: Routines and Hexadecimal PC Values

```
$ oct-top /tftpboot/traffic-gen --address
Function Name:             Total Hits
fvwrite_r:                      1
0x000000001014586c:             1
cvmx_sysinfo_get:               1
0x000000001003ddf4:             1
memmove:                        1
0x0000000010138070:             1
cvmx_scratch_read64:            5
0x0000000010009730:             5
cvmx_get_cycle:                14
cvmx_clock_get_count:          14
0x000000001001ab38:            14
cvmx_send_single:             107
cvmx_write64_uint64:          107
0x0000000010009758:           107
fastpath_receive:             289
0x000000001000791c:             4
0x0000000010007730:            22
0x00000000100077c8:            17
0x00000000100077a0:            14
0x0000000010007794:            13
0x00000000100077b0:            13
0x00000000100077d4:            13
0x0000000010007764:            11
0x000000001000775c:            10
0x0000000010007778:             7
0x0000000010007788:             7
0x00000000100077ac:             7
0x00000000100078e0:             7
0x0000000010007734:             6
0x00000000100077d8:             6
0x0000000010007740:             5
```

B.3.6 Top of Selected Cores

Run `top` utility on selected cores instead of on all cores running the application.

Table B–12 oct-top: Running utility on selected Cores

```
$ oct-top /tftpboot/traffic-gen --core_mask=0x23
Function Name:                    Total Hits
cvmx_srio_initialize:                 1
kuseg - 32bit User space:             1
cvmx_spinlock_lock:                 152
cvmx_spinlock_unlock:               164
process_cmd_packetio:               358
periodic_update:                    529
packet_transmitter:                1752
main:                              2882
statistics_gatherer:               2882
fastpath_receive:                  5790
```

B.3.7 OCTEON Hardware Statistics

Display OCTEON hardware statistics that include TADs, DRAM info, FPA pools, and SSO I/O queues.

Table B–13 OCTEON Hardware Statistics

```
$ oct-top -s
Core counters:
brmis 63426[0], 32526698[1], 32527062[2], 32528303[3], 32528310[4], 32565565[5],
total (excluding 0) brmis 162675938,
sync 0[0], 0[1], 0[2], 0[3], 0[4], 0[5],
total (excluding 0) sync 0,

L2 statistics for TAD 0:
bus_xmc(addr) count:                   22
bus_xmd(store) count:                  19
bus_rsc(commit) count:                 22
bus_rsd(fill) count:                   22
bus_ioc(IO req) count:                 22
bus_ior(IO req) count:                 22
hit count:                          39650
miss count:                             0
lfb-wait-vab bus utilization:          0%
lfb-wait-lfb bus utilization:          0%

DRAM Statistics:
DRAM ops count:                 142357388
dclk count:                 3564342216764
utilization:                        0.00%

FPA Pool statistics:
FPA Pool 0 (Packet Buffers):     816 free
FPA Pool 1 (WQE buffers):        490 free
FPA Pool 2 (PKO cmd buffers):      0 free, PKO output queues 128

POW/SSO Qos Input Queue Statistics
Queue 0: WQE count  0, Buffers allocated  0
Queue 1: WQE count  0, Buffers allocated  0
Queue 2: WQE count  0, Buffers allocated  0
Queue 3: WQE count  0, Buffers allocated  0
Queue 4: WQE count  0, Buffers allocated  0
Queue 5: WQE count  0, Buffers allocated  0
Queue 6: WQE count  0, Buffers allocated  0
Queue 7: WQE count  0, Buffers allocated  0
Total WQEs in all QOS levels: 0 (in-unit & in-memory)
Total Free Buffers in all QOS levels: 243
```

B.3.8 Cores Running Simple Executive from Linux

Run `oct-top` on the cores running Simple Executive application booted from Linux.

Table B–14 oct-top: Simple Executive from Linux

Boot linux on first core

```
OCTEON ebb6300# tftpboot 0 vmlinux.64
OCTEON ebb6300# bootoctlinux 0 coremask=0x1 mem=512m

$ oct-app-boot namedblock coremask=0x3e
   CVMX_SHARED: 0x10190000-0x101a0000
   Active coremask = 0x1
   INFO: Launching application on coremask 0xff0
   Available Coremask = 0x0
$ oct-top named-block –core_mask=0x3e
```

B.3.9 Frequently Called Routines

List the frequently called routines for Linux (works without passing `vmlinux` file).

Table B–15 Frequently-called Routines

```
$ oct-top -k
Function Name:                   Total Hits
__rcu_process_callbacks:             1
cpumask_next_and:                    1
dequeue_entity:                      1
find_busiest_group:                  1
find_next_bit:                       1
tg_shares_up:                        1
64bit User space:                 2719
r4k_wait:                        13589
```

B.3.10 Refresh Rate

Control the refresh rate of the top output.

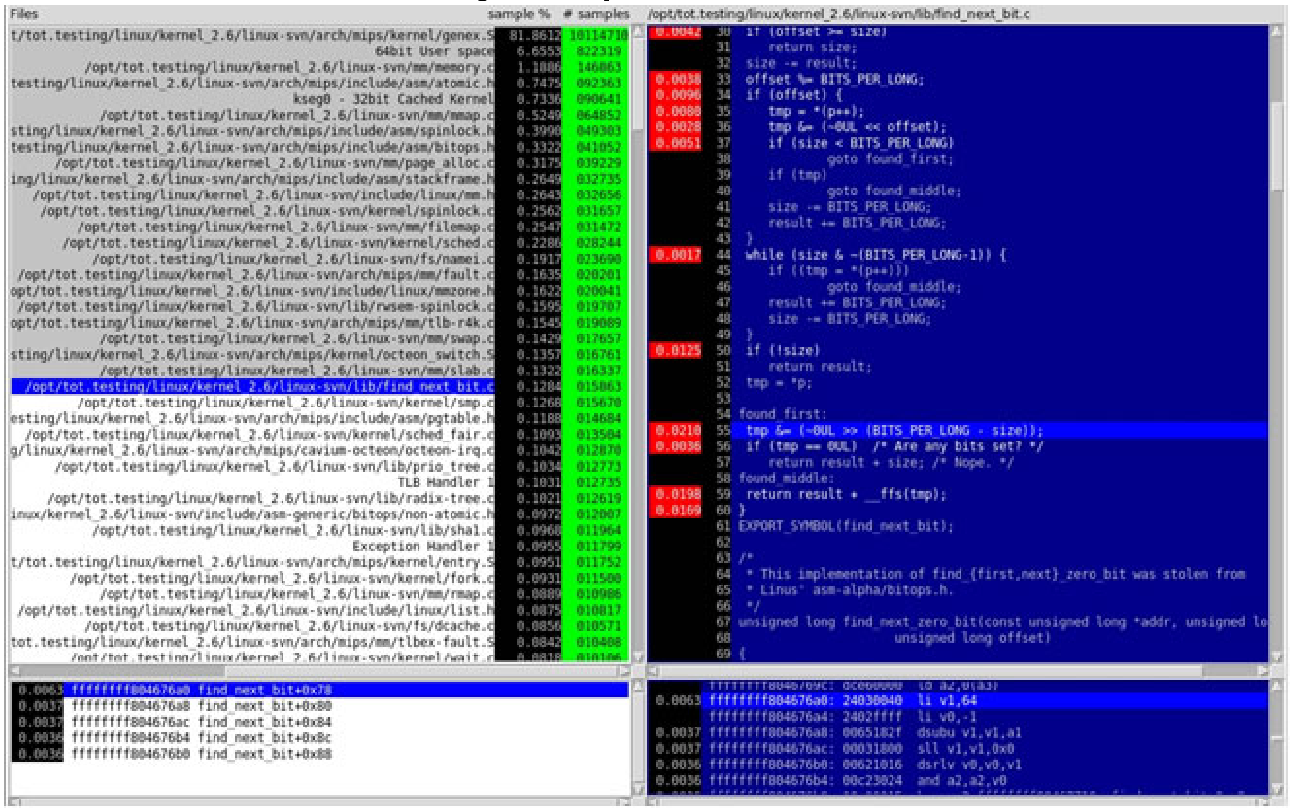
Table B–16 oct-top: Refresh rate

```
$ oct-top /tftpboot/-gen --dwell=3
```

B.4 `perfzilla`

B.4.1 Understanding `perfzilla` output

Figure B–1 perfzilla GUI



Files
sample %
# samples
/opt/tot.testing/linux/kernel_2.6/linux-svn/lib/find_next_bit.c

```
t/tot.testing/linux/kernel_2.6/linux-svn/arch/mips/kernel/genex.S  81.8612  10114710
                                                  64bit User space   6.6553    822319
       /opt/tot.testing/linux/kernel_2.6/linux-svn/mm/memory.c   1.1806    146863
testing/linux/kernel_2.6/linux-svn/arch/mips/include/asm/atomic.h   0.7475    092363
                                         kseg0 - 32bit Cached Kernel   0.7336    090641
          /opt/tot.testing/linux/kernel_2.6/linux-svn/mm/mmap.c   0.5249    064852
sting/linux/kernel_2.6/linux-svn/arch/mips/include/asm/spinlock.h   0.3990    049303
testing/linux/kernel_2.6/linux-svn/arch/mips/include/asm/bitops.h   0.3322    041052
     /opt/tot.testing/linux/kernel_2.6/linux-svn/mm/page_alloc.c   0.3175    039229
ing/linux/kernel_2.6/linux-svn/arch/mips/include/asm/stackframe.h   0.2649    032735
   /opt/tot.testing/linux/kernel_2.6/linux-svn/include/linux/mm.h   0.2643    032656
    /opt/tot.testing/linux/kernel_2.6/linux-svn/kernel/spinlock.c   0.2562    031657
        /opt/tot.testing/linux/kernel_2.6/linux-svn/mm/filemap.c   0.2547    031472
      /opt/tot.testing/linux/kernel_2.6/linux-svn/kernel/sched.c   0.2286    028244
          /opt/tot.testing/linux/kernel_2.6/linux-svn/fs/namei.c   0.1917    023690
  /opt/tot.testing/linux/kernel_2.6/linux-svn/arch/mips/mm/fault.c   0.1635    020201
opt/tot.testing/linux/kernel_2.6/linux-svn/include/linux/mmzone.h   0.1622    020041
  /opt/tot.testing/linux/kernel_2.6/linux-svn/lib/rwsem-spinlock.c   0.1595    019707
opt/tot.testing/linux/kernel_2.6/linux-svn/arch/mips/mm/tlb-r4k.c   0.1545    019089
          /opt/tot.testing/linux/kernel_2.6/linux-svn/mm/swap.c   0.1429    017657
sting/linux/kernel_2.6/linux-svn/arch/mips/kernel/octeon_switch.S   0.1357    016761
          /opt/tot.testing/linux/kernel_2.6/linux-svn/mm/slab.c   0.1322    016337
  /opt/tot.testing/linux/kernel_2.6/linux-svn/lib/find_next_bit.c   0.1284    015863
        /opt/tot.testing/linux/kernel_2.6/linux-svn/kernel/smp.c   0.1268    015670
esting/linux/kernel_2.6/linux-svn/arch/mips/include/asm/pgtable.h   0.1188    014684
  /opt/tot.testing/linux/kernel_2.6/linux-svn/kernel/sched_fair.c   0.1093    013504
g/linux/kernel_2.6/linux-svn/arch/mips/cavium-octeon/octeon-irq.c   0.1042    012870
    /opt/tot.testing/linux/kernel_2.6/linux-svn/lib/prio_tree.c   0.1034    012773
                                                     TLB Handler 1   0.1031    012735
   /opt/tot.testing/linux/kernel_2.6/linux-svn/lib/radix-tree.c   0.1021    012619
inux/kernel_2.6/linux-svn/include/asm-generic/bitops/non-atomic.h   0.0972    012007
          /opt/tot.testing/linux/kernel_2.6/linux-svn/lib/sha1.c   0.0968    011964
                                               Exception Handler 1   0.0955    011799
t/tot.testing/linux/kernel_2.6/linux-svn/arch/mips/kernel/entry.S   0.0951    011752
       /opt/tot.testing/linux/kernel_2.6/linux-svn/kernel/fork.c   0.0931    011500
          /opt/tot.testing/linux/kernel_2.6/linux-svn/mm/rmap.c   0.0889    010986
   /opt/tot.testing/linux/kernel_2.6/linux-svn/include/linux/list.h   0.0875    010817
         /opt/tot.testing/linux/kernel_2.6/linux-svn/fs/dcache.c   0.0856    010571
tot.testing/linux/kernel_2.6/linux-svn/arch/mips/mm/tlbex-fault.S   0.0842    010408
       /opt/tot.testing/linux/kernel_2.6/linux-svn/kernel/wait.c   0.0818    010106
```

```
0.0042  30  if (offset >= size)
        31      return size;
        32  size -= result;
0.0038  33  offset %= BITS_PER_LONG;
0.0096  34  if (offset) {
0.0088  35      tmp = *(p++);
0.0028  36      tmp &= (~0UL << offset);
0.0051  37      if (size < BITS_PER_LONG)
        38              goto found_first;
        39      if (tmp)
        40              goto found_middle;
        41      size -= BITS_PER_LONG;
        42      result += BITS_PER_LONG;
        43  }
0.0017  44  while (size & ~(BITS_PER_LONG-1)) {
        45      if ((tmp = *(p++)))
        46              goto found_middle;
        47      result += BITS_PER_LONG;
        48      size -= BITS_PER_LONG;
        49  }
0.0125  50  if (!size)
        51      return result;
        52  tmp = *p;
        53
        54 found_first:
0.0210  55  tmp &= (~0UL >> (BITS_PER_LONG - size));
0.0036  56  if (tmp == 0UL)  /* Are any bits set? */
        57      return result + size; /* Nope. */
        58 found_middle:
0.0198  59  return result + __ffs(tmp);
0.0169  60 }
        61 EXPORT_SYMBOL(find_next_bit);
        62
        63 /*
        64  * This implementation of find_{first,next}_zero_bit was stolen from
        65  * Linus' asm-alpha/bitops.h.
        66  */
        67 unsigned long find_next_zero_bit(const unsigned long *addr, unsigned lo
        68                          unsigned long offset)
        69 {
```

```
0.0063  ffffffff804676a0 find_next_bit+0x78
0.0037  ffffffff804676a8 find_next_bit+0x80
0.0037  ffffffff804676ac find_next_bit+0x84
0.0036  ffffffff804676b4 find_next_bit+0x8c
0.0036  ffffffff804676b0 find_next_bit+0x88
```

```
0.0063  ffffffff804676a0: 24030040  li v1,64
        ffffffff804676a4: 2402ffff  li v0,-1
0.0037  ffffffff804676a8: 0065182f  dsubu v1,v1,a1
0.0037  ffffffff804676ac: 00031800  sll v1,v1,0x0
0.0036  ffffffff804676b0: 00621016  dsrlv v0,v0,v1
0.0036  ffffffff804676b4: 00c23024  and a2,a2,v0
```

B.4.2 Source Code Window

The Source Window display sample %, line number, and the C source code line, as shown in the following table:

Table B–17 perfzilla: Source Code Window

```
0.0210       55       tmp &= (~0UL >> (BITS_PER_LONG – size))
```

B.4.3 Function List window

The Function List window displays the % sample, respective PC value, and function name (including offset), as shown in the following table:

Table B–18 perfzilla: Function List Window

```
0.0063 ffffffff804676a0 find_next_bit+0x78
0.0037 ffffffff804676a8 find_next_bit+0x80
0.0037 ffffffff804676ac find_next_bit+0x84
0.0036 ffffffff804676b4 find_next_bit+0x8c
0.0036 ffffffff804676b0 find_next_bit+0x88
```

B.4.4 ASM window

The ASM window displays information about the disassembly of the selected C source code, as show in the following table:

Table B–19 perfzilla: ASM Window

```
0.0063    ffffffff804676a0: 24030040    liv1,64
          ffffffff804676a4: 2402ffff    liv0,-1
0.0037    ffffffff804676a8: 0065182f    dsubuv1,v1,a1
0.0037    ffffffff804676ac: 00031800    sllv1,v1,0x0
0.0036    ffffffff804676b0: 00621016    dsrlvv0,v0,v1
0.0036    ffffffff804676b4: 00c23024    anda2,a2,v0
0.0036    ffffffff804676b8: 10c00015    beqza2,ffffffff80467710
<find_next_bit+0xe8>
```

The ASM window updates information about the disassembly of the selected C source code in the source file, as shown in the following table:

Table B–20 perfzilla: ASM Window - Source Code Disassembly

```
tmp &= (~0UL >> (BITS_PER_LONG – size))
li  v1,64
     store the value of BITS_PER_LONG in v1 register
li  v0,-1
     store ~0UL value in to v0 register
dsubu  v1,v1,a1
     v1 = v1 – a1 (v1 is 64 and a1 refers to the subroutine find_next_bit argument
size)
sll  v1,v1,0x0
dsrlv  v0,v0,v1
     v0 = ~0uL >> (BITS_PER_LONG - size)
and  a2,a2,v0
     a2 = a2 & v0
```