NAME

cpc − The Cell-Perf-Counter tool

SYNOPSIS

cpc [options] [workload]

DESCRIPTION

The cpc tool is used for setting up and using the hardware performance counters in the Cell Broadband Engine processor. These counters allow a user to see how many times certain hardware events are occurring, which is useful while analyzing the performance of software running on a Cell system.

CPC has two basic modes of operation. The first is workload mode, which allows the user to specify another program to run while the counters are active. CPC sets up the desired events, starts the counters, then runs the specified program. When the workload is complete, cpc stops the counters, reads and displays the counter values, and resets the performance monitoring unit. In this mode the counters will only run on the CPU where the workload is running, and only while the workload is active. This provides a very accurate view of the performance of that single process.

The second mode is system-wide mode. In this mode, the user specifies the events and options, along with an optional set of CPUs and time duration. Cpc starts the counters on all specified CPUs and lets them run for the specified amount of time. If no time is given, cpc will run the counters until an EOF (control-d) is received. The counters will monitor all processes running on all specified CPUs, and display the results after the counters have been stopped.

OPTIONS

The following options can be used with cpc:

-h, --help
Display this summary of options.

-v, --verbose
Verbose output.

-V, --version
Display the version number of this tool.

Specifying Events To Count

-l, --list-events
Display all available events.

-e, --events EVENT,EVENT,...
Comma-separated list of events to count. Events can be specified using their name or number. See **--list-events** for a full listing of available events. There are four counters available, and events are assigned to counters in the order that they are specified. To skip a counter, leave it blank (two consecutive commas).

The counters and list of events are reset before loading this list of events. Any events specified on a previous run of cpc are deleted.

Multiple --events options can be given. Each one is treated as a separate "event-set" and all event-

sets are loaded into the kernel. The kernel will run each one for a specific amount of time (known as the "switch-timeout") and then rotate to the next set in round-robin order. After running the last set, it rotates back to the first set again and continues the loop. The switch-timeout can be given on a per-set basis along with the specification of events, or a global switch-timeout value can be given with the --switch-timeout option.

Each EVENT specification should be in the form:0
[-+]<name|number>[.<C|E>][:subunit][=initial_value]

Events are counted either in cycles or edges. An event that is counted in cycles will give the number of cycles that elapsed while the event was occurring. An event that is counted in edges will give the number of times that the event occurred, regardless of how long it took for it to occur.

By default, an event is counted on it’s "positive polarity". By prefixing the event name/number with a hyphen, the event will be counted on it’s "negative polarity". For cycle-counting events, this means the counter will increment for every cycle when the event is *not* occurring. For edge-counting events, this means the counter will increment *after* the event has completed, instead of immediately upon the event occurring.

While most events are counted only in cycles or only edges, some events allow the user to decide which way to count. In these cases, the count type is specified by appending a ".C" or a ".E" to the event number or name. The events that allow this choice are shown with the appended ".C" or ".E" in the list displayed with the **--list-events** option. The default is ".C".

When counting an event in the SPU or MFC groups, the desired SPU/MFC (subunit) number is given by suffixing the event name/number with a colon and the SPU/MFC number. If no number is given, it defaults to 0. The subunit number is ignored for other event groups.

The counters are initialized to 0 unless an initial value is given with "=VALUE" after the event number or name.

To count all clock cycles without regard to events, specify event "C".

To count events on the external trigger, specify event "Ext". This can also be prefixed with "+" or "-" to indicate the desired polarity, and suffixed with ".C" or ".E" to indicate whether cycles or edges should be counted.

To specify a switch-timeout for an individual event-set, add:
,{switch_timeout_value} to the end of the list of events. See the information for the --switch-timeout option for allowable timeout values. If no timeout value is given for an event-set, the global value given by the --switch-timeout option is used.

Events are grouped together according to logic units within the Cell processor. The PMU can only count events for up to two groups at once, but can count any number of events within a group (up to the number of counters). Use the **--list-events** option to get a list of all events and which groups they belong to. Counting the same event for two different SPUs/MFCs counts as two different groups.

Due to hardware limitations, not all combinations of events can be counted at the same time. Use the **--query** option to verify whether a particular combination is valid without actually counting anything.

Example:
--events 2104=501,-IL1_Miss_Cycles_t1.E,C,4104:2=344

Counter 0 counts event 2104, beginning
with an initial value of 501.

Counter 1 counts the IL1_Miss_Cycles_t1 event
(number 2123) with negative polarity and an
initial value of zero. It counts the number
of events that occur (instead of the number of
cycles elapsed while the event is occurring).

Counter 2 counts all processor cycles.

Counter 4 counts event 4104 for SPU 2,
beginning with an initial value of 344.

-s, --switch-timeout TIME

Global timeout value for switching event-sets in the kernel. If a timeout value is not given for an individual event-set, this value is used. Suffix the value with 'n' for nanoseconds, 'u' for microseconds, 'm' for milliseconds, or 's' for seconds. If no suffix is given, the value is in milliseconds. The default timeout value is 50 milliseconds. This option only applies if multiple event-sets are specified - no switching occurs if only one event-set is used.

-c, --cpus CPUS

Monitor the specified physical CPUs (CBE nodes) in system wide mode instead of monitoring a single workload. 'CPUS' can be 'all' to monitor all available CPUs, or it can be a comma-seperated list of physical CPUs. This option only applies if a workload is not specified.

Note that this option takes *physical* CPU numbers, since there is only one PMU per physical CPU. Each physical CPU has two hardware threads, and each of these threads is reported by Linux as a separate logical processor. In a dual-CPU system, physical CPU 0 will contain logical CPUs 0 and 1, and physical CPU 1 will contain logical CPUs 2 and 3.

-t, --time TIME

Monitor the CPUs (specified with the --cpus option) in system-wide mode for the specified number of seconds. If --cpus is given, but --time isn't, the tool will start monitoring and wait until an EOF (control-D) is received. This option only applies in system-wide monitoring mode - it is ignored when monitoring a specific workload.

-m, --mode MODE

Count only when in specified MODE:

a count in all modes
s count in supervisor (kernel) mode
p count in problem (user) mode
h count in hypervisor mode
k alternate for s
u alternate for p
y alternate for h

Miscellaneous Actions

-q, --query
Query to see if an event combination is valid without performing any actions.

--single-threaded
Don’t use multiple threads for running the Perfmon2 system-calls in parallel to all the CPUs. Make the system-calls sequentially from the main cpc process. This should only be necessary for debugging the Perfmon2 kernel driver.

Output Formatting Options

-o, --output filename
Write the normal text output to the specified file instead of to the screen.

-H, --html filename
Write output in HTML format to the specified file. If --list-events is also given, the output file will contain an HTML version of the event list.

--xml filename
Write output in XML format to the specified file. If --list-events is also given, the output file will contain an XML version of the event list.

-x, --hex
Display counter values in hexidecimal.

--hw-signal-names
Display hardware signal names instead of descriptive event names.

Hardware Sampling Options

-i, --interval INTERVAL
Sampling INTERVAL. Suffix the value with ’n’ for nanoseconds, ’u’ for microseconds, or ’m’ for milliseconds. If no suffix is given, the value is in core clock cycles. The minimum value is 10 cycles or 4 nanoseconds. The counters are reset at the beginning of each interval. If this option is specified, the sampling data will be stored in the hardware trace-buffer and the sampling mode will default to "count" (unless **--sampling-mode** option specifies an alternate mode).

--sampling-mode TYPE
Indicates the TYPE of data stored to the sampling-buffer. If "none", the counters contain data for the entire duration of the workload. If the "--interval" option is specified, the default is "count".
none Store no data. (default)
count Store counter values.
occurrence Store occurrence data.
threshold Store threshold data.

--sampling-buffer-size SIZE_FACTOR
Multiple of the hardware trace-buffer size to use for the in-kernel sampling-buffer. The hardware trace-buffer is 16 kB, and the default SIZE_FACTOR value is 10, which gives a default kernel sampling-buffer size of 160 kB.

--thermal-data
Collect thermal data in the sampling-buffer. This option cannot be used with occurrence or threshold sampling. It also cannot be used when using counters two or three. This option can also be used without giving a set of events to count.

--overwrite-samples
Overwrite the hardware sampling data when the buffer fulls up. Only the data from the most recent 1024 samples will be available.

--ppu-branch-trace HW_THREADS
Enable PPU branch address tracing for the specified hardware threads on all CPUs that are being monitored. 'HW_THREADS' can be 'all' to monitor all hardware threads, or it can be a comma-seperated list of hardware threads. This option requires that one of the hardware-sampling modes be enabled. Counter 0 cannot be used with this option.

--ppu-bookmark-trace HW_THREADS
Enable PPU bookmark tracing for the specified hardware threads on all CPUs that are being monitored. 'HW_THREADS' can be 'all' to monitor all hardware threads, or it can be a comma-seperated list of hardware threads. This option requires that one of the hardware-sampling modes be enabled. Counter 0 cannot be used with this option.

Start/Stop Qualifier Options

--start-on-ctr0 N
Start counters when counter 0 has counted "N" events. Counter 4 cannot be used with this option enabled, unless **--stop-on-ctr4** is also used. "N" must be limited to a 16-bit value.

--stop-on-ctr4 N
Stop counters when counter 4 has counted "N" events. Counter 0 cannot be used with this option enabled, unless **--stop-on-ctr0** is also used. "N" must be limited to a 16-bit value.

--start-on-trigger0
Start counters upon debug bus trigger 0.

--start-on-trigger1
Start counters upon debug bus trigger 1.

--start-on-trigger2
Start counters upon debug bus trigger 2.

--start-on-trigger3
Start counters upon debug bus trigger 3.

--stop-on-trigger0
Stop counters upon debug bus trigger 0.

--stop-on-trigger1
Stop counters upon debug bus trigger 1.

--stop-on-trigger2
Stop counters upon debug bus trigger 2.

--stop-on-trigger3
Stop counters upon debug bus trigger 3.

--start-on-ppu-spr-trigger1
Start counters upon PPU SPR trigger 1.

--stop-on-ppu-spr-trigger2
Stop counters upon PPU SPR trigger 2.

--start-on-event1
Start counters upon debug bus event 1.

--stop-on-event2
Stop counters upon debug bus event 2.

--start-on-ppu-th0-bookmark
Start counters upon PPU hardware-thread 0 bookmark start.

--start-on-ppu-th1-bookmark
Start counters upon PPU hardware-thread 1 bookmark start.

--stop-on-ppu-th0-bookmark
Stop counters upon PPU hardware-thread 0 bookmark stop.

--stop-on-ppu-th1-bookmark
Stop counters upon PPU hardware-thread 1 bookmark stop.

--restart-enable
Allows prequalifier start after prequalifier stop.

Workload
Program and arguments to execute while counting events. The --cpus and --time options are ignored if a workload is specified.

If no workload is specified and the --cpus option is not given, all CPUs will be monitored in system-wide mode. Additionally, if the --time option is not given, the tool will start monitoring and wait until control-D (EOF) is received.

HARDWARE SAMPLING

The Cell PMU provides a mechanism for the hardware to periodically read the counters and store the results in a hardware buffer. This allows the cpc tool to collect a large number of counter samples while greatly reducing the number of calls that cpc has to make into the kernel.

Use the **--interval** option to specify the length of the sampling period, in time or in clock cycles. In most cases, the counters should be initialized to zero, since they are all reset to their initial value at the end of

each sampling period. As it runs and monitors the workload or the system, cpc periodically reads the contents of the in-kernel sampling-buffer. Information for all accumulated samples is displayed after monitoring is complete.

In addition to simply sampling the counters, the Cell PMU provides a variety of other sampling modes.

Occurrence sampling (**--sampling-mode occurrence**) monitors up to 64 events at once. Each sample contains one bit for each event, indicating whether the event occurred at least once during that interval. All of the events in an event-group are monitored, and up to two groups can be monitored at once. Use the **--events** option to specify any event in the group(s) to be sampled.

Threshold sampling (**--sampling-mode threshold**) monitors the counters, and records one bit for each active counter that indicates whether the associated event occurred at least some specified number of times (the "threshold"). For this to be useful, the events need to be initialized to the desired "threshold" value. If no initial value is given for an event, a threshold of zero is used, which means that event will always hit its threshold.

Thermal sampling (**--thermal-data**) collects information about the temperature in various areas of the Cell processor. The PPU and all eight SPUs contain temperature sensors, and the values of these sensors can be routed to the sampling-buffer. This option cannot be used with occurrence or threshold sampling, since the data is stored in the same place in the sampling-buffer. This option can be used with regular count sampling, but cannot be used when counters 2, 3, 6, or 7 are enabled. This option can also be used by itself, without loading any other events to count.

Partial samples

When cpc is running in one of the hardware-sampling modes, kernel events can occur that may disrupt the collection of a sample. These kernel event include context-switching the workload process that is being monitored, switching event-sets (if multiple sets of events are given), and handling PMU interrupts. When one of these occur, the hardware may be part-way through collecting the current sample. However, the hardware sampling-buffer cannot be restored to a previous state, so the kernel must record the data that is available for that fraction of a sample. When the PMU is enabled again, it will start collecting a new sample.

In regular counter-sampling mode, the kernel has access to the hardware counters that are being sampled, so the intermediate counter values can be copied into the sampling-buffer. In the cpc output, these samples will indicate they are only partially complete, and give the percentage of the interval that was completed before it was interrupted.

However, in occurrence, threshold and thermal sampling modes, the kernel does not have access to the hardware where the intermediate values are stored during each interval. In these cases, the cpc output will show all zero data for that sample, and will mark the line as incomplete.

PPU branch-address tracing

CPC provides the ability to gather a trace of certain branch addresses that occur on the PPU. In particular, the bclr(l), bcctr(l), and rfid instructions, which are generally used for subroutine return, can be traced for either or both of the PPU hardware-threads. Use the --ppu-branch-trace option to enable this feature, and specify the list of hardware-threads that should be monitored, or ’all’ to monitor all hardware-threads.

This feature can be used by itself, or in addition to any of the other hardware- sampling modes. If this is used with any other hardware-sampling mode, the PPU branch addresses will be interspersed with the counter sampling data.

PPU bookmark tracing

In addition to tracing branch addresses, CPC can generate traces for writes to the PPU bookmark special-purpose registers. When the --ppu-bookmark-trace option is used, all values written to the bookmark register will be copied into the sampling buffer. Just like branch-address tracing, bookmark tracing can be used at the same time as one of the hardware-sampling modes, and the bookmark values will be interspersed with the counter sampling data.

From the command line, to write to the bookmark register for the first physical CPU, echo the desired value into the /sys/devices/system/cpu/cpu0/pmu_bookmark file. For the second physical CPU, use the file /sys/devices/system/cpu/cpu2/pmu_bookmark. These files can also ben written to from within another program by using open() and write(). However, the value written to the file must be an ASCII string of the desired value, not the actual numerical value.

Hardware-sampling kernel module

In order to use any of these hardware sampling options, the **perfmon_cell_hw_smpl** kernel module must be loaded. If cpc is running as root, it will automatically load this module if it isn't already loaded. If cpc is running as a non-root user, the module must already be loaded. On Redhat and Fedora-based systems, the module can be autoloaded at boot-time by adding the file **/etc/sysconfig/modules/perfmon.modules**, with execute permissions, containing the following two lines:

```
#!/bin/sh
modprobe perfmon_cell_hw_smpl > /dev/null 2>&1
```

START/STOP QUALIFIERS

CPC provides several options for starting and/or stopping the counters after some condition has occurred. When a --start-on options is used, the counters will not actually begin counting until the related event occurs. When a --stop-on options is used, the counters will stop once the related event occurs.

Start on counter 0 and stop on counter 4

CPC provides options to start and stop the counters when a certain number of countable events have occurred. In order to use this feature, specify the --start-on-ctr0 and/or --stop-on-ctr4 option. Each of these options takes an argument which is the number of events to count before starting/stopping the counters. When either of these options are used, the other counter (0 or 4) cannot be used as a regular counter. Both counters can be used if both --start-on-ctr0 and --stop-on-ctr4 are used. In addition, the argument to these options must be limited to a 16-bit value (less than 65536).

Start/stop on PPU bookmark writes

CPC provides options to start and stop the counters whenever a value is written to one of the PPU bookmark registers. Each physical Cell CPU has two logical CPUs (a.k.a. hardware threads), and each logical CPU has its own bookmark register. If the --start-on-ppu-th0-bookmark option is used, writes to the first hardware thread on any monitored physical CPU will start the counters. If the --start-on-ppu-th1-bookmark option is used, writes to the second hardware thread on any monitored physical CPU will start the counters. The semantics are the same for the --stop-on-ppu-th0-bookmark and --stop-on-ppu-th1-bookmark options.

To write to the bookmark register, use the files /sys/devices/system/cpu/cpuX/pmu_bookmark, where 'X' is the logical CPU number. Logical CPUs 0 and 1 are the two hardware threads on physical CPU 0, and logical CPUs 2 and 3 are the two hardware threads on physical CPU 1. From the command line, echo a decimal value into the file to write to the bookmark register. These files can also be written to from within another program by using open() and write(). However, the value written to the file must be an ASCII string of the

desired value, not the actual numerical value.

Specific values must be written to the bookmark register in order to start and/or stop the counters. To start the counters, use the value 2^63, or 9223372036854775808. To stop the counters, use the value 2^62, or 4611686018427387904.

Start/stop qualifier restrictions

The start/stop qualifiers go into effect each time the PMU is enabled. However, in order to support features like per-process monitoring, multiple event-sets, hardware-sampling and 64-bit virtual counters, the kernel frequently disables and re-enables the PMU. For example, when a monitored process is context switched out, the PMU is disabled and the PMU state is saved in the process context. When that monitored process is scheduled on the CPU again, the PMU state is reloaded and the PMU is enabled again. However, the PMU treats that as a whole new monitoring session with regard to the start/stop qualifiers, instead of a continuation of a previous session. Even if a start/stop qualifier had been triggered previously, it will wait for that qualifier to occur again before the counters actually start.

Therefore, all the start/stop qualifier options are only supported in system- wide mode, with a single event-set, and without any hardware-sampling modes. In addition, the 64-bit virtual counters are disabled. If a counter reaches its maximum value (2^32-1), it will stop counting.

EXAMPLES

Example 1: Workload mode, single event-set

cpc --events 2104.E,2123.E math_test 1000 100000

Run the program "math_test 1000 100000", and measure events 2104 and 2123 (L1 i-cache misses for hardware threads 0 and 1) while the command is running.

Output:

The cpc output always starts with a header and a copy of the cpc command that was run. When running in workload mode, the header info is followed by basic information about the workload that was monitored. Then the actual results are displayed. In workload mode, there will be one set of results for each event-set given on the command-line. The results include the values of all the PMU control registers that were used for that event-set, followed by data for each hardware counter. Each counter line shows the count value, the number and name of the event that was assigned to that counter, whether the event was counted in 'events' (E) or 'cycles' (C), and which SPU was monitored (if applicable for that event).

```
********************************
*  Cell Perf-Counter Results  *
********************************

Command: cpc --events 2104.E,2123.E math_test 1000 100000

Workload
--------
  Command:     math_test 1000 100000
  Process ID:  24459
  Return value: 0
  Started:     Tue Aug  7 12:40:14 2007
  Stopped:     Tue Aug  7 12:40:17 2007
  Duration:    2.925406 seconds
```

```
Results For Workload
====================

  Event-Set 0
  -----------

    Control Registers
    -----------------

      Register          Value
      --------         ----------
      group_control      0x00000000
      debug_bus_control  0x00000000
      trace_address      0x00000000
      ext_tr_timer       0x00000000
      pm_status          0xc0000000
      pm_control         0x000c0000
      pm_interval        0x00000000
      pm_start_stop      0x00000000

      pm0_control        0x11400000
      pm1_control        0x5d400000

      pm0_event          0x00000000fffff7c8
      pm1_event          0x00000000fffff7b5

    Data Counters
    -------------

                    Event   Count
    Ctr      Count  Number  Type   SPU   Event Name
    ---  ---------- ------  -----  ---  ----------
     0         43754   2104   E          IL1_Miss_Cycles_t0
     1         30995   2123   E          IL1_Miss_Cycles_t1
```

Example 2: System-wide mode, single event-set

cpc --events C,2100,2119 --cpus all --time 10s

Measure clock-cycles and branch instructions committed on both hardware threads, for all processes on all CPUs for ten seconds.

Output:

In system-wide mode, a set of results is displayed for each physical CPU that was given by the --cpus option, and each of these sets will contain one set of results for each event-set. The format of the results is the same as in workload mode.

```
********************************
*  Cell Perf-Counter Results  *
********************************

Command: cpc --events C,2100,2119 --cpus all --time 10s
```

```
Results For Physical CPU 0
==========================

  Event-Set 0
  -----------

    Control Registers
    -----------------

      Register          Value
      --------         ----------
      group_control     0x00000000
      debug_bus_control 0x00000000
      trace_address     0x00000000
      ext_tr_timer      0x00000000
      pm_status         0xe0000000
      pm_control        0x000c0000
      pm_interval       0x00000000
      pm_start_stop     0x00000000

      pm0_control       0x42c00000
      pm1_control       0x01c00000
      pm2_control       0x4dc00000

      pm0_event         0x0000000000000000
      pm1_event         0x00000000fffff7cc
      pm2_event         0x00000000fffff7b9

    Data Counters
    -------------

                        Event    Count
      Ctr      Count    Number   Type    SPU   Event Name
      ---  -----------  ------  -----  ---  ----------
       0   32010733502       0    C          System Clock Cycles
       1       2482563    2100    C         Branch_Commit_t0
       2       1191865    2119    C         Branch_Commit_t1

Results For Physical CPU 1
==========================

  Event-Set 0
  -----------

    Control Registers
    -----------------

      Register          Value
      --------         ----------
      group_control     0x00000000
      debug_bus_control 0x00000000
      trace_address     0x00000000
      ext_tr_timer      0x00000000
      pm_status         0xe0000000
```

```
  pm_control       0x000c0000
  pm_interval      0x00000000
  pm_start_stop    0x00000000

  pm0_control      0x42c00000
  pm1_control      0x01c00000
  pm2_control      0x4dc00000

  pm0_event        0x0000000000000000
  pm1_event        0x00000000fffff7cc
  pm2_event        0x00000000fffff7b9

Data Counters
-------------

                    Event   Count
 Ctr      Count     Number  Type   SPU   Event Name
 ---  -----------  ------  -----  ---  ----------
  0   32010774708       0   C          System Clock Cycles
  1       1216673    2100   C         Branch_Commit_t0
  2       1177159    2119   C         Branch_Commit_t1
```

Example 3: Workload mode, multiple event-sets, additional output formats

```
cpc --events 2111,2130 --events 2205,2221 --switch-timeout 100ms \
   --output test1.txt --xml test1.xml --html test1.html \
   math_test 1000 100000
```

Run the "math_test" program, and count PPC instructions committed in one event-set, and L1 d-cache load misses in a second event-set. Run each event-set for 100 milliseconds before switching to the other event-set. Write the text output to the file test1.txt, and generate html and xml outputs in the files test1.html and test1.xml.

Output:

The output file test1.txt will contain the following text, with one set of results for each of the event-sets measured. The test1.html file will contain the results in very similar looking format for viewing in a web browser, and test1.xml will contain data in a format that can be input into VPA.

```
********************************
*  Cell Perf-Counter Results  *
********************************

Command: cpc --events 2111,2130 --events 2205,2221 --switch-timeout 100ms --output test1.txt --xml test1.xml --ht

Workload
--------
  Command:     math_test 1000 100000
  Process ID:  24838
  Return value: 0
  Started:     Tue Aug  7 13:10:18 2007
  Stopped:     Tue Aug  7 13:10:21 2007
  Duration:    2.925257 seconds
```

Results For Workload
====================

Event-Set 0

Control Registers

Register	Value
group_control	0x00000000
debug_bus_control	0x00000000
trace_address	0x00000000
ext_tr_timer	0x00000000
pm_status	0xc0000000
pm_control	0x000c0000
pm_interval	0x00000000
pm_start_stop	0x00000000
pm0_control	0x2dc00000
pm1_control	0x79c00000
pm0_event	0x00000000fffff7c1
pm1_event	0x00000000fffff7ae

Data Counters

Ctr	Count	Event Number	Type	SPU	Event Name
0	1463270	2111	C		PPC_Commit_t0
1	334414208	2130	C		PPC_Commit_t1

Event-Set 1

Control Registers

Register	Value
group_control	0x00000000
debug_bus_control	0x00000000
trace_address	0x00000000
ext_tr_timer	0x00000000
pm_status	0xc0000000
pm_control	0x000c0000
pm_interval	0x00000000
pm_start_stop	0x00000000
pm0_control	0x15c00000
pm1_control	0x55c00000

```
  pm0_event         0x00000000fffff763
  pm1_event         0x00000000fffff753

 Data Counters
 -------------

                    Event    Count
  Ctr      Count     Number   Type    SPU   Event Name
  ---  -----------  ------   -----   ---   ----------
   0     830462939    2205     C           DL1_Miss_t0
   1             0    2221     C           DL1_Miss_t1
```

Example 4: Hardware sampling

cpc --events C,2104,2123 --interval 1ms math_test 1000 100000

Run the "math_test" program and collect counter samples every 1 millisecond.

Output:

In all hardware sampling modes, the results for each event-set include a short table showing which events are assigned to each counter, followed by a (potentiall very long) table of the raw sampling data. Each counter has one column of data, and each row begins with the sample number. As explained in the "Partial Samples" section above, some of the lines of data include a note at the end indicating that the line represents an incomplete sample, and gives the percentage complete for the interval. The sampling-buffer header information gives the total number of samples as well as the number of incomplete samples.

```
********************************
*  Cell Perf-Counter Results  *
********************************

Command: cpc --events C,2104,2123 --interval 1ms math_test 1000 100000

Workload
--------
  Command:      math_test 1000 100000
  Process ID:   25216
  Return value: 0
  Started:      Tue Aug  7 13:48:36 2007
  Stopped:      Tue Aug  7 13:48:39 2007
  Duration:     2.930169 seconds

Results For Workload
====================

  Event-Set 0
  -----------

    Control Registers
    -----------------

      Register          Value
      --------         ----------
```

```
  group_control      0x00000000
  debug_bus_control  0x00000000
  trace_address      0x00000000
  ext_tr_timer       0x00000000
  pm_status          0x00400000
  pm_control         0x500c0000
  pm_interval        0xffcf2bff
  pm_start_stop      0x00000000

  pm0_control        0x42c00000
  pm1_control        0x11c00000
  pm2_control        0x5dc00000

  pm0_event          0x0000000000000000
  pm1_event          0x00000000fffff7c8
  pm2_event          0x00000000fffff7b5

Sampling Buffer
---------------

  Interval: 3200000 clock cycles
  Number of Samples: 2926 (10 incomplete)

        Bit    Event    Count
  Ctr   Width  Number   Type    SPU   Event Name
  ---   -----  ------   -----   ---   ----------
   0     32        0    C           System Clock Cycles
   1     32     2104    C             IL1_Miss_Cycles_t0
   2     32     2123    C             IL1_Miss_Cycles_t1

  Sample     Ctr 0       Ctr 1       Ctr 2
  ------  -----------  -----------  -----------
      0     1122916          0      316640   (only 35.1% complete)
      1      692188          0      277432   (only 21.6% complete)
      2       13850          0        1732   (only 0.4% complete)
      3     1005070          0      307826   (only 31.4% complete)
      4     3200000          0      516100
      5     3200000      67272       59396
      6     3200000          0           0
      7     3200000          0           0
      8     3200000          0           0
      9     3200000       1066        9304
     10     3200000          0           0

  [ example data removed for brevity ]

   2915     3200000        518         962
   2916     3200000          0           0
   2917     3200000          0           0
   2918     3200000          0           0
   2919     3200000        720         572
   2920     3200000          0           0
   2921     3200000          0           0
   2922     3200000          0           0
```

```
2923     3200000       626        824
2924     2829438         0     460496   (only 88.4% complete)
2925      197044     92618          0   (only 6.2% complete)
```

Example 5: Hardware sampling, threshold mode

```
cpc --events 2104=100,2123=200 --interval 1ms \
   --sampling-mode threshold math_test 1000 100000
```

Run the "math_test" program and collect threshold samples every 1 millisecond. The threshold for event 2104 is 100, and the threshold for event 2123 is 200.

Output:

In the sampling data results, each sample shows a 1 or 0 for each counter to indicate whether that counter exceeded its threshold value during that interval.

```
********************************
*  Cell Perf-Counter Results  *
********************************

Command: cpc --events 2104=100,2123=200 --interval 1ms --sampling-mode threshold math_test 1000 100000

Workload
--------
  Command:     math_test 1000 100000
  Process ID:   25323
  Return value: 0
  Started:     Tue Aug  7 13:58:18 2007
  Stopped:     Tue Aug  7 13:58:21 2007
  Duration:    2.926313 seconds

Results For Workload
====================

  Event-Set 0
  -----------

    Control Registers
    -----------------

      Register          Value
      --------         ----------
      group_control     0x00000000
      debug_bus_control 0x00000000
      trace_address     0x00000000
      ext_tr_timer      0x00000000
      pm_status         0x00400000
      pm_control        0x700c0000
      pm_interval       0xffcf2bff
      pm_start_stop     0x00000000

      pm0_control       0x11c00000
      pm1_control       0x5dc00000
```

```
  pm0_event       0x00000000fffff7c8
  pm1_event       0x00000000fffff7b5

Threshold Sampling Buffer
-------------------------
Counters that overflowed during the sample interval.

  Interval: 3200000 clock cycles
  Number of Samples: 2992 (112 incomplete)

       Threshold    Event   Count
  Ctr     Value     Number  Type   SPU   Event Name
  ---  ----------  ------  -----  ---  ----------
   0        100    2104    C           IL1_Miss_Cycles_t0
   1        200    2123    C           IL1_Miss_Cycles_t1

       Counters
       --------
  Sample  0  1
  ------ -- --
     0  0  0 (Incomplete)
     1  0  0 (Incomplete)
     2  0  0 (Incomplete)
     3  0  0 (Incomplete)

  [ skipped for brevity ]

    278  0  0
    279  1  1
    280  1  0
    281  1  0
    282  1  0
    283  1  1
    284  0  0
    285  0  0
    286  0  0
    287  1  1
    288  0  0
    289  0  0
    290  0  0
    291  1  1
    292  0  0
    293  0  0
    294  0  0
    295  1  1
    296  0  0
    297  0  0
    298  0  0
    299  1  1
    300  0  0
```

SEE ALSO

oprofile(1)

AUTHOR

Kevin Corry <kevcorry@us.ibm.com>