EMON

User's Guide

Copyright © 2013–2017 Intel Corporation

All Rights Reserved

Revision: 2.2

Legal Information

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© 2017 Intel Corporation.

Revision History

Revision Number	Description	Revision Date
1.0	Initial release.	January 2013
2.0	Completed major documentation updates and renamed to *EMON User's Guide*. Added Installation and Examples chapters. Updated examples in section 2.2 General Options: Collection.	February 2016
2.1	Added and removed event modifiers.	February 2017
2.2	Updated SEP driver version.	April 2017

Contents

1 About EMON

EMON is a low-level command line tool that provides the ability to specify performance events and attributes, allocate and configure the event resources, and retrieve event counts in processors and chipsets. EMON V9 now shares its driver code base with the SEP 3 tool, whereas the previous version (EMON V8) had a completely separate code base.

1.1 Intended Audience

Use this document if you use EMON to analyze performance data on your system.

1.2 Conventions and Symbols

The following conventions are used in this document.

Table 1 Conventions and Symbols used in this Document

`This type style`	Indicates an element of syntax, reserved word, keyword, filename, computer output, or part of a program example. The text appears in lowercase unless uppercase is significant.
This type style	Indicates the exact characters you type as input. Also used to highlight the elements of a graphical user interface such as buttons and menu names.
`This type style`	Indicates a placeholder for an identifier, an expression, a string, a symbol, or a value. Substitute one of these items for the placeholder.
`[ items ]`	Indicates that the items enclosed in brackets are optional.
`{ item | item }`	Indicates to select only one of the items listed between braces. A vertical bar (|) separates the items.
... (ellipses)	Indicates that you can repeat the preceding item.

2 Installation

2.1 Windows*

1. Create a directory where you want to install EMON. For example: `C:\Program Files (x86)\IntelSWTools\emon`.
2. Unzip the EMON install package (`EMON_Windows.zip`) to this directory.

   The following directories should appear in the specified location:

   ```
   bin32
   bin64
   config
   docs
   sep_vars.cmd
   ```
3. Open a command prompt as an Administrator.
4. Install and register the sampling driver from the `bin32` directory using the following command: `sepreg.exe -i`
5. Run `sep_vars.exe` to set up the environment variables for the current shell.
6. Start EMON using the following command: `emon -v`
7. Stop the EMON driver using the following command: `net stop sepdrv<version>`

   For example, the following command is used for EMON version 3.15: `net stop sepdrv3_15`

2.2 Linux*

1. Create a directory where you want to install EMON. For example: `/opt/intel/emon`
2. Copy the EMON install package (`EMON_<platform>.tgz`) to this directory.
3. Unpack the EMON install package. For example: `tar -xjzf EMON_<platform>.tgz`

   The following directories should appear in the specified location:

   ```
   bin32
   bin64
   config
   docs
   lib32
   lib64
   sepdk
         include
         src
   sep_vars.sh
   ```
4. Navigate to the sepdk directory: `cd /opt/intel/emon/sepdk`

5. Build the EMON driver using the following command: `./build-driver -ni`

   You may be prompted for the root password.
6. Install the EMON driver using the following command: `./insmod-sep4 -g [user_group]`
7. Navigate to the directory from which you want to run the EMON command line tool.
8. Create a bash shell: `sh`
9. Set up the EMON runtime environment by sourcing the `sep_vars.sh` file in the current bash shell: `source /opt/intel/emon/sep_vars.sh`
10. Start EMON using the following command: `emon -v`

3 Usage

Start the EMON command line tool using the following syntax: `emon -v`

Use the EMON command line tool with the following syntax:

```
emon [general-options] -C (event-definitions) [application-command-line]
```

3.1 General Options: Information

-h

Display help information

-1

List the event names that can be monitored on the host platform.

-? | -H

Print detailed information about the events that can be monitored on the host platform.

-! <event name>

Print detailed information about <event name>.

--dry_run

Print the event sets that will be used during a collection given the current list of events. For example, the following command:

```
emon --dry-run -C (BR_MISSP_EXEC, CPU_CLK_UNHALTED.CORE_P,
CPU_CLK_UNHALTED.NO_OTHER, CPU_CLK_UNHALTED.BUS)
```

would produce the following output:

```
Event Set 0
  BR_MISSP_EXEC
     CPU_CLK_UNHALTED.CORE_P
Event Set 1
     CPU_CLK_UNHALTED.NO_OTHER
     CPU_CLK_UNHALTED.BUS
```

The given events are scheduled into two separate event sets due to the number of general purpose counters that are available.

-M

Print the OS processor to logical/physical hardware processor mapping.

-v

Display version information about EMON and the host platform's processor(s).

3.2 General Options: Collection

-C <event1,event2,...>

Specify one or more events for which the performance data will be collected. Events should be separated by a comma.

For example, the following command:

```
emon -t4 -C (BR_MISSP_EXEC, CPU_CLK_UNHALTED.CORE_P)
```

will output the following:

```

BR_MISSP_EXEC 11,170,797,057  1,182,076     945,618
CPU_CLK_UNHALTED.CORE_P 11,170,797,057      271,588,482      224,067,845
```

The event behavior can be modified for the core events by including an event modifier. See the Core Event Modifiers section for more details.

-l <loops>

The number of times each event set is monitored. Default value is 1. Event sets are interleaved. For example, if 2 events sets A and B are specified, and time equals 4 and loops equal 2, event set A is monitored for 4 seconds, then event set B is monitored for 4 seconds, then event set A is monitored for 4 seconds, and finally event set B is monitored for 4 seconds.

For example, the following command:

```
emon -t4 -l2 -C (BR_MISSP_EXEC, CPU_CLK_UNHALTED.CORE_P)
```

will output the following:

```

BR_MISSP_EXEC 11,170,797,057  1,182,076     945,618
CPU_CLK_UNHALTED.CORE_P 11,170,797,057      271,588,482      224,067,845
==========
BR_MISSP_EXEC 11,171,812,071  1,427,945     1,336,304
CPU_CLK_UNHALTED.CORE_P 11,171,812,071      363,409,158      309,894,165
==========

8.000s real    0.070s user  7.929s system
```

-L <time>

Range for random delay of the monitor interval. Specified in seconds. A random delay of 0 to <time> is introduced between each sample. When used, each monitor interval is the value of the -t switch plus the random delay between 0 and <time> milliseconds. Defaults to 0. This functionality will be automatically disabled if -t switch is set to 0.

For example, given the following command:

```
emon -t1 -L0.5 -l5 -C CPU_CLK_UNHALTED.REF
```

the output will be 5 intervals, each varying between 1 second and 1.5 seconds in length, like below:

```
CPU_CLK_UNHALTED.REF 3,183,859,770   228,832,296   193,736,770
==========
CPU_CLK_UNHALTED.REF 3,785,798,247   213,735,732   200,822,149
==========
CPU_CLK_UNHALTED.REF 3,938,517,954   450,800,763   392,327,029
==========
CPU_CLK_UNHALTED.REF 4,036,692,576   252,970,557   211,929,868
==========
CPU_CLK_UNHALTED.REF 3,938,388,479   701,150,656   350,296,989
==========
6.765s real    0.078s user  6.687s system
```

-s <delay>

One time delay in seconds before monitoring is started.

-t <time>

Time (seconds) that an event set is monitored. Default value is 3. A value of 0 allows monitoring to last until the application exits.

-w

Limit loops. The number of loops is limited by the application's execution time. For example, if the total monitoring time specified by the time and loop switches is greater than the actual application execution time, the collection is stopped after the application exits.

3.3 General Options: Input/Output

-c

Print system time (i.e. date-time) for each time interval. It is only available in the Standard and Quiet output modes.

-d

Results are printed in formatted decimal. Formatted decimal includes comma separators. Formatted decimal is the default.

-f <output file>

Emon output is written to <output file>. The -f switch creates a new output file.

-F <output file>

Emon output is appended to <output file>. If <output file> doesn't exist, it will be created.

-i <input file>

Emon command line arguments are provided by <input file>. Comments are indicated with a '#'. All text following a '#' in an input file is ignored.

-n

Print wall clock, user, and system time for each time interval. It is only available in the Standard and Quiet output modes.

-q

Quiet mode output. Minimal information is output.

-u

Results are printed in unformatted decimal. Unformatted decimal does not include comma separators.

-x

Results are printed in hex with a leading '0x'.

-X

Spreadsheet-friendly Format. The results are output in tab-separated format.

3.4 Core Event Modifiers

The following table lists the event modifiers and provides a short description of each modifier.

Table 2 Core Event Modifiers

Modifier	Description
`:amt<0/1>`	Sets (1) or clears (0) the event's Any Thread control bit. A value of 0 sets the Any Thread bit to 0, which causes the event to be counter on a per logical processor basis when applicable. A value of 1 sets the Any Thread bit to 1, which causes the event to be counted on a per core basis. The default Any Thread value is defined by the executing core's associated event list.
`:c <cmask>`	Specifies the mask value that will be compared to the events count of the detected microarchitectural condition by the logical processor during a single cycle. If the event

	count is greater than or equal to this mask, the counter is incremented by one. Otherwise the counter is not incremented. The value must be in the range of 0x0 to 0xff. The default <cmask> value is defined by the executing core's associated event list.
`:e<0/1>`	Enables edge detection of the selected microarchitectural condition when set. The logical processor counts the number of deasserted to asserted transitions for any condition that can be expressed by other fields. A value of 0 disables edge detect and a value of 1 enables edge detect. The default edge detect value is defined by the executing core's associated event list. ***NOTE:*** Edge detection is enabled only when CMASK and ANYTHR flags are set. For example, `emon -l1 -t0.1 -C "MACHINE_CLEARS.COUNT, MACHINE_CLEARS.COUNT:amt1:e1:c1"`
`:i<0/1>`	When the invert flag is set, inverts :c <cmask> comparison, so that both greater than or equal to and less than comparisons can be made (<0>: greater than comparison, <1>: less than comparison). If : c <cmask> is programmed to 0, invert flag is ignored. A value of 0 disables invert and a value of 1 enables it. The default invert value is defined by the executing core's associated event list.
`:u<umask>`	<umask> indicates the value of the event's unit mask. The <umask> value must be in the range 0x0 to 0xff. Note that predefined events often include a non-zero umask value. Their umask values can be overridden by including a :u event modifier in the event's specification. The default <umask> value is defined by the executing core's associated event list.
`:USER`	Specifies that events are counted only when the processor is operating at privilege levels 1, 2, or 3. This flag can be used in conjunction with the SUP flag.
`:SUP`	Specifies that events are counted only when the processor is operating at privilege level 0. This flag can be used in conjunction with the USER flag.
`:ALL`	Specifies that events are counted when the processor is operating at all privilege levels.
`:cp`	In Check Point – When this modifier is specified, the data result will not include counts that occurred inside of an aborted TSX region.
`:tx`	In Transaction – When this modifier is specified, the data result will only include counts that occurred inside a TSX region, regardless of whether that region was aborted or committed.

4 Examples

4.1 Basic Command Example

The most basic EMON command is:

```
emon -C (CPU_CLK_UNHALTED.REF)
```

Processor-specific events can be found with the -1 (the number one) flag.

If not otherwise specified, EMON will monitor one 3 second interval. To change either the interval length or the number of intervals (or loops), the user can use the –t or –l options, respectively. The following output is produced with the basic EMON command. The first column has the event name, the second column contains the clock ticks followed by the counts for the event per processor.

```
CPU_CLK_UNHALTED.REF    6,100,381,287     1,052,111,151     887,635,455
==========
3.000s real       0.062s user    2.937s system
```

The basic command creates the data output in quiet mode, which means a minimal amount of output. To print out the headers for importing into a spreadsheet, specify the spreadsheet mode with the –X flag.

```
Sample     Clocks     CPU_CLK_UNHALTED.REF[CPU0] CPU_CLK_UNHALTED.REF[CPU1]
1   6,100,381,287     1,052,111,151     887,635,455
```

4.2 Additional Examples

1. Collection with one group, 3 second interval (total runtime: 3 seconds):

```
emon -C "CPU_CLK_UNHALTED.REF"
```

2. Collection with one group, 0.5 second interval, 5 loops (total runtime: 2.5 seconds):

```
emon -l5 -t0.5 -C "CPU_CLK_UNHALTED.REF"
```

3. Collection with one group of 2 events, 0.5 second interval, 5 loops (total runtime: 2.5 seconds):

```
emon -l5 -t0.5 -C "CPU_CLK_UNHALTED.REF,INST_RETIRED.ANY"
```

4. Collection with two groups, 0.5 second interval, 5 loops (total runtime: 5 seconds):

```
emon -l5 -t0.5 -C "CPU_CLK_UNHALTED.REF;INST_RETIRED.ANY"
```

5. Collection with one group, running <an_app>. Data collection will terminate when app exits:

```
emon -t0 -C "CPU_CLK_UNHALTED.REF,INST_RETIRED.ANY" <an_app>
```