

PPC750 Performance Monitor User's Guide

Donald Meyer

February 18, 2003



1 Introduction

The PPC Performance Monitor Program, a.k.a. “Perfmon” is provided to obtain per-task performance statistics for applications running under the Wind River VxWorks real-time operating system on the PPC750 architecture. Perfmon is compiled and linked into a VxWorks loadable module. When loaded with user modules, it is available for use through Perfmon commands, without any modification to user code and at negligible performance penalty.

User-specified statistics are generated and written for user-specified time intervals to a user-specified file or to the serial-port.

2 PerfMon Commands

The commands for operating the PerfMon application would typically be entered individually or as a script to the Wind River Tornado WindSh. Alternately, commands could be entered to a “tip” or “telnet” window connected to the target PPC750’s serial port. This however creates an intermingling of user command input and Perfmon output.

Perfmon statistics output is expected to appear on the PPC750 target’s serial port. However, error messages resulting from bad command input appears in the WindSh, if WindSh is in use.

Commands `mon_Install`, `mon_Stop`, and `mon_Select` may be called from user code if `<mon_Init.h>` is `#include’d`, and commands `mon_Run` and `mon_Namefile` may be called if `<mon_TaskDisplay.h>` is `#include’d`.

2.1 mon_Install Command

This command is used to activate a previously loaded Perfmon executable module. It has no parameters. It must be successfully issued before any other Perfmon command is given. In addition to initializing data structures and counters, `mon_Install` performs the following actions.

- Attaches the task hook-functions that are called upon task creation, switching, and deletion. This is fundamental to the Perfmon design, and as far as is known is a feature unique to vxWorks.
- Creates watchdog timer to drive the periodic data collection and display.
- Spawns the Perfmon print-task (`tMonPrint`) and idle-task (`tMonIdle`).
- Connects the interrupt service routine generated by the rollover of any of the four PPC750 Performance Counter Registers.
- Sets the PPC750 Performance Control Registers to record the measurements indicated by the `mon_Select` command with parameter value 0.

- Sets the PPC750 Performance Control Registers to enable interrupts from the rollover of any of the four PPC750 Performance Counter Registers.

Tasks spawned prior to issuing the `mon_Install` command will be included in the list of tasks for which statistics are collected and displayed. A limit of 500 tasks overall is imposed, with 200 of these permitted prior to the `mon_Install` command. A task which was deleted during a data collection interval will be reported, but be unnamed.

If a subsequent `mon_Install` command is issued without target processor reboot, unpredictable results are to be expected. There is not at present a `mon_Uninstall` command.

2.2 `mon_Select` Command

This command is used to specify the type of performance information to be captured and displayed. There are only four performance monitoring registers in the PPC750 architecture, each of which can support as many as 16 counters, each for a particular architecture-defined measurement. Since a given “statistic” often requires two measurements, there can be and are conflicts in what can be captured at the same time. In particular, the L1 I-cache (Instruction Cache) and L1 D-cache (Data Cache) miss rates cannot be obtained simultaneously.

The user does not select individual counters, but rather one of three fixed sets of “statistics” as specified by the “`display_mode`” parameter to this command.

For each time interval, for each task, the percentage time and the true MIPS are always displayed. The `mon_Select` command specifies what further statistics are presented. MIPS (Millions of Instructions Per Second) is calculated from the count of executed (not dispatched) instructions over the collection interval.

This command can be issued anytime after a successful `mon_Install`. If the new selection matches the current selection, a message is output and no other action is performed. An invalid selection (not 0, 1, or 2) produces a message, and no other action is performed.

The starting value is “`display_mode`” 0.

Command Format:

`Mon_Select (int display_mode)`

2.2.1 For mon_Select display_mode value 0

Display for each task

$$\text{L1 I-cache miss rate} = \frac{\text{Number of L1 I-cache misses} * 100}{\text{Number of instructions executed}}$$

$$\text{Dispatch Ratio} = \frac{\text{Number of dispatched instructions}}{\text{Number of instructions executed}}$$

Task Stack High Water Mark as a percentage. This is the only statistic, which is not presented on a per-interval basis. It is the largest value recorded since the tMonIdle task was spawned by mon_Install. tMonIdle must run for the task stack high water mark to be valid.

.... in the following fashion:

PER-TASK STATISTICS

Interval: Start at 5.000, Duration of 5.000 seconds.

Tasks: End Total: 16 Max Total: 16 Created: 0 Deleted: 0
Swaps/Sec: 41.6

Task Name	%Time	MIPS	L1 I-Cache Miss %	Dispatch Ratio	%Stack Usage
tExcTask	0.000	0.000	0.000	0.00	11.72
tLogTask	0.000	0.000	0.000	0.00	5.78
tAioWait	0.000	0.000	0.000	0.00	1.95
tAioIoTask1	0.000	0.000	0.000	0.00	0.87
tAioIoTask0	0.000	0.000	0.000	0.00	0.87
tNetTask	0.026	75.515	0.876	1.29	6.97
tRlogind	0.000	0.000	0.000	0.00	24.08
tTelnetd	0.000	0.000	0.000	0.00	20.39
tPortmapd	0.000	0.000	0.000	0.00	11.72
tWdbTask	0.000	0.000	0.000	0.00	18.50
tShell	0.000	0.000	0.000	0.00	5.67
tTelnetOutTask	0.005	70.721	2.919	1.26	26.00
tTelnetInTask	0.000	0.000	0.000	0.00	28.09
tMonPrint	0.012	176.908	0.423	1.30	61.43
tMonIdle	95.055	150.044	0.000	1.00	11.74
tMonAppl	4.901	109.391	0.000	1.00	5.45
Total	100.000	148.032	0.000%	1.00	

2.2.2 For mon_Select display_mode value 1

Display for each task

$$\text{L1 D-cache miss rate} = \frac{\text{Number of L1 D-cache misses} * 100}{\text{Number of load and store instructions}}$$

$$\text{Data Ratio} = \frac{\text{Number of load and store instructions}}{\text{Number of instructions executed}}$$

.... in the following fashion:

PER-TASK STATISTICS

Interval: Start at 0.000, Duration of 7.992 seconds.

Tasks: End Total: 16 Max Total: 16 Created: 0 Deleted: 1
Swaps/Sec: 37.4

Task Name	%Time	MIPS	L1 D-Cache Miss %	Data Ratio
tExcTask	0.001	64.827	1.622	0.42
tLogTask	0.000	0.000	0.000	0.00
... etc ...				
tMonPrint	0.000	0.000	0.000	0.00
tMonIdle	95.074	150.015	3.192	0.50
tMonAppl	4.903	109.435	16.051	0.20
	0.000	0.000	0.000	0.00
	0.000	40.227	3.430	0.45
Total	100.000	148.006	3.383%	0.49

2.2.3 For mon_Select display_mode value 2

Display for each task

$$\text{L1 Cast-out Ratio} = \frac{\text{Number of L1 D-cache cast-outs}}{\text{Number of L1 D-cache misses}}$$

.... in the following fashion:

```
PER-TASK STATISTICS
Interval: Start at 7.988, Duration of 4.000 seconds.
Tasks: End Total: 16 Max Total: 16 Created: 0 Deleted: 0
Swaps/Sec: 39.3

Task Name          %Time    MIPS    L1 Cast_out
                  Ratio
tExcTask           0.000    0.000    0.000
tLogTask           0.000    0.000    0.000
... etc ...
tMonPrint          0.010    168.215    0.036
tMonIdle           95.066    150.043    0.010
tMonAppl           4.898    109.436    0.914
Total              100.000    148.037    0.074%
```

2.3 mon_Namefile Command

Perfmon produces output to file or stdout, i.e. the serial-port, but not to both simultaneously. The text report is identical for each case. The issuance of a mon_Namefile command (usually) causes the output to be directed to a named file. This file is created if necessary, and then opened for appending. The file contents are not visible until the file is closed. It remains open until closed by one of the following methods:

- Issuing a mon_Stop command (parameter value not 0).
- Issuing another mon_Namefile command specifying a file name.
- Issuing another mon_Namefile command specifying a file name of "0".

Note that the "user" doing the file writing is vxWorks. This means that the Perfmon user's file directory must have permissions set to allow VxWorks to write to it. The resulting file can be copied or deleted by the Perfmon user, but may not be written by the Perfmon user. If only a file name, and not a full path, is specified in mon_Namefile, then the file written will appear in a VxWorks directory.

Command Format:

`mon_Namefile (char const* pfilename, int show_progress)`

Parameter “pfilename“ is usually a quoted string specifying the file’s full pathname. If the file does not exist, it is created. In any event the file is opened for appending.

If the parameter “pfilename“ is 0, Perfmon output is henceforth sent to the serial port, and any open Perfmon output file is closed. If a `mon_Namefile` with “pfilename“ of 0 is commanded while writing to file, the file is closed and a switch is made to serial-port output without a break in the sequence of time intervals.

Parameter “show_progress“ is given a non-zero value if it is desired to display a one-line report to the serial-port at the end of each interval. This is useful since the file is written invisibly, and it may be useful to know the progress so far, and that all is proceeding as expected. If “show_progress“ is zero, then no one-line reports are generated.

The one-line report has the following form:

```
File Record: Start at 15.000, Duration of 5.000 seconds.
```

2.4 mon_Run Command

This command initiates a data collection session. A session consists of the collecting and subsequent display of statistics (as specified by `mon_Select`) over specified time intervals. The statistics displayed for an interval pertain to that time interval only (except for the task stack high water mark).

The intervals are contiguous, and the session continues until terminated by a `mon_Stop` command. The final interval will likely be truncated, depending upon when the `mon_Stop` is received. The values displayed for a truncated interval are valid. The interval time can be changed in the course of a session, with a resulting truncated interval as the transition is made.

Command Format:

```
mon_Run (int interval_time)
```

interval_time in integer seconds. A minimum value of 2 seconds is imposed.

The interval time appears in a banner as “Duration”, as follows:

```
Interval: Start at 5.997, Duration of 6.000 seconds.
```

2.5 mon_Stop Command

This command terminates a data collection session, which consists of the collecting and subsequent display of statistics (specified by mon_Select) over specified time intervals. The session continues until terminated by a mon_Stop command. The final interval will likely be truncated, at a time that depends upon when the mon_Stop is received. The values displayed for a truncated interval are valid.

Command Format:

```
mon_Stop (int file_control)
```

“file_control“ is a flag controlling the following behavior:

If the display output is going directly to stdout, i.e. the serial-port, then this flag has no effect. Serial-port output is indicated until such time that a mon_Namefile command is issued.

If a mon_Namefile command (with non-0) filename has been issued, then the designated file is written, and the serial-port is not. In this case a mon_Stop “file_control“ parameter value of 0 will terminate the session as in the serial-port case, leaving the file open for further appended writing. Since the file is not closed, its contents are not visible.

For a “file_control“ parameter value of not 0, the session is terminated as in the serial-port-case and the file is closed. This makes the file visible, containing the output for one or more sessions. The display mode then reverts to serial-port display until such time that another mon_Namefile command is issued.

3 Example User Input

The following could have been entered to a Wind River Tornado WindSh, or to a “tip” or “telnet” window connected to the target PPC750’s serial port. Three “sessions” are indicated, one printing to serial-port followed by two printing to the same file.

```
-> cd "/home/dmeyer/x2k/osa/mon/c++"  
-> ld < mon_load.ghs  
-> mon_Install  
-> mon_Run 20  
-> mon_Stop  
-> mon_Namefile "/home/dmeyer/x2k/osa/mon/c++/files/fileA",1  
-> mon_Select 1  
-> mon_Run 8  
-> mon_Stop  
-> mon_Select 2  
-> mon_Run 30  
-> mon_Stop 1
```

4 Limitations

- The mon_Install command must be successfully issued before any other Perfmon command is given.
- A limit of 500 tasks overall is imposed, with 200 of these permitted prior to the mon_Install command. These limits are easily changed in source code.
- The Perfmon print-task (tMonPrint) runs at the (low) priority of 240, and should be adjusted as necessary (in source code) if user tasks would prevent it from running.
- The Perfmon idle-task runs (tMonIdle) at the lowest priority of 255. This should not conflict with a user idle-task running at the same priority. However, the Perfmon idle-task must be allowed to run if valid task stack high water mark statistics are to be collected.
- There is not at present a mon_Uninstall command.
- A minimum value of 2 seconds is imposed for the statistics collection and display time interval.

5 Motivation

The development of Perfmon was motivated by the attributes of the RAD750 space flight computer developed by BAE Systems, Manassas, Virginia. This implementation of the PPC750 architecture includes Level-1 (L1) data and instruction cache, but not Level-2 cache. This makes the overall runtime performance particularly sensitive to cache utilization.