PMON - Performance Monitoring Utility V2.2
Users Guide
10/15/96

Additional copies of this document or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

 CG-041493

1. Objective:

PMON is a performance monitoring tool for Intel processors that retrieves information about a sequence of code that is running under Windows* 95. This information may then be used for various purposes, including performance tuning, performance validation, code coverage, and system tuning.

2. System Requirements:

- Windows* 95
- Intel Pentium® processor, Pentium Pro processor, or Pentium processor with MMX™ technology.

3. PMON Installation:

Create a new directory (e.g. : C:\PMON) and unzip the PMON files into it. The executables are:

PM32_APP.EXE
PM32_DLL.DLL
PMON.VXD

The zipped file may also include an additional file :

EVENTS.INI

Note: If this file is not included, it will be created by the Pmon Utility.

4. Main operation modes:

Figure 1 illustrates the basic components of PMON. These are described in detail below.

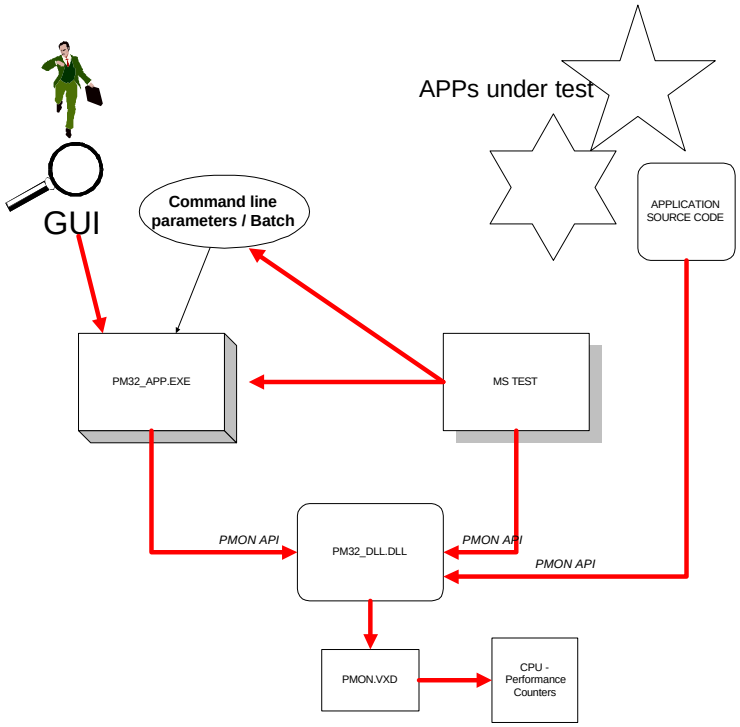

Figure 1

4.1 User interface

In this mode, referred to hereafter as “Pmon Utility”, you operate the utility interactively while running the application you want to test. Refer to Figure 2 for the features described below.

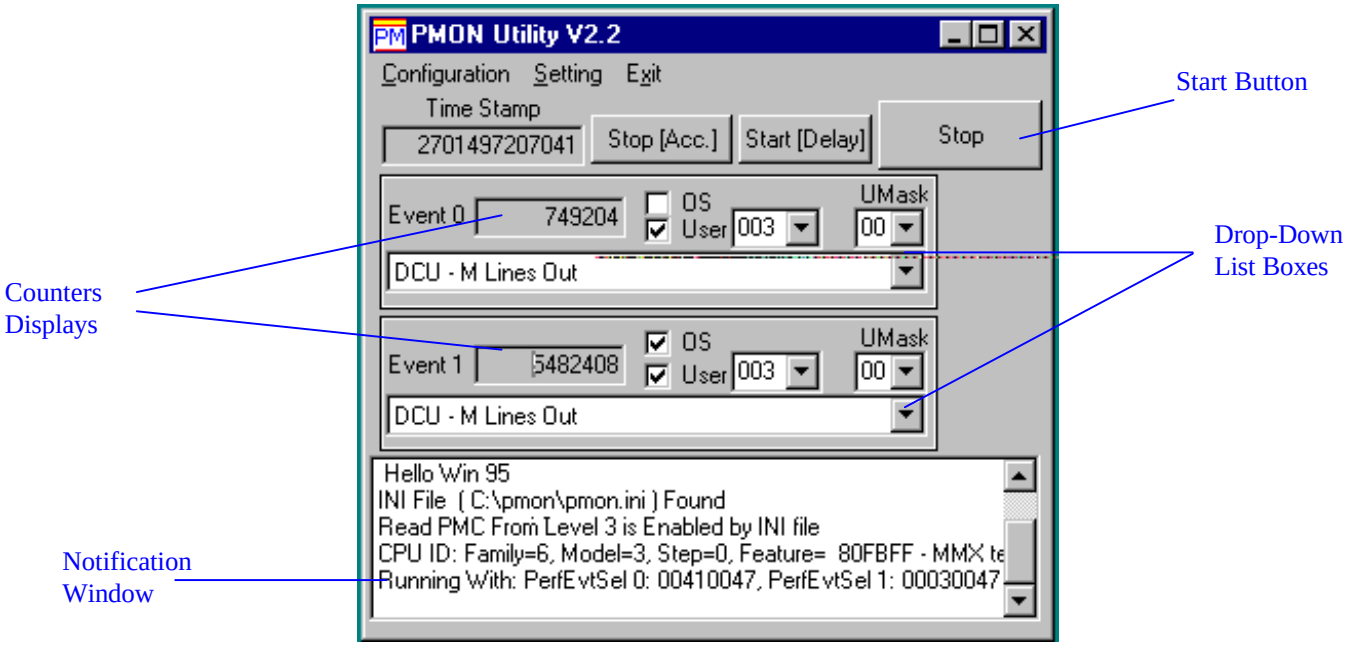

Figure 2

First, select the events you want to measure through the two drop-down list boxes, and then start the counters by pushing the “Start” button. The “Start” button will now display “Stop.” You will see the counters updated every second. When you hit the “Stop” button, the summary of this measurement will be displayed in the notification window. You can then copy these messages to the Windows* clipboard or to a file.

There are several options for starting/stopping the counters, and for logging the results into a file. For details, see the explanations later in this guide. Also, the source code of this utility is supplied with this package and may be used for reference or modified as needed.

4.2 Command line interface

In this mode you activate the PMON utility along with some command line parameters. The command line parameters allow you to activate the utility, select the events, start and stop the counters, and log the results to a file. Command line implementation can be done by running it from a Windows 95 DOS* box, by creating and editing the properties of the utility’s desktop icon or by using a test tool utility such as Microsoft* Visual Test.
Note: Each time you call this utility it loads again and executes.

4.3 Programming interface

In this mode, referred to hereafter as “Pmon API”, you interface with the DLL APIs from your code or test tool utility. Loading, initializing the DLL, starting/stopping and retrieving the information is done by the user from the application code. This is most accurate and non-intrusive mode, however it requires some programming and you must have the source code of the application under test. This mode can also be used with a test tool utility.

5. PMON Features

Table 1 lists the features of PMON, which are described in detail below.

Feature	User / Command line interface	Programming/API interface
Event selection	Yes	Yes
Start/Stop counting	Yes	Yes
Delayed start	Yes	Yes (User implements)
Accumulate start	Yes	Yes (User implements)
Stop counting and log	Yes	Yes
Install new events	Yes (events.ini)	Yes
Read counters from ring3	Yes	Yes
Send event directly to counters control registers	No	Yes

Table 1

5.1 Current Files and version numbers:

PM32_APP.exe	The application	- V2.2
PM32_DLL.dll	The DLL	- V2.2
PMON.vxd	The VxD	- V2.2

Warning: All version numbers must be as listed above or Pmon will not work correctly. You can check the DLL and VxD version numbers by looking at the first two lines in the Notification window. The utility checks the version numbers and displays a warning in the notification window if they do not match.

5.2 The VxD

The VxD is dynamically loaded the first time you activate the DLL and unloaded when you exit. Through the API or via the command line parameter you have the option to specify to unload the VxD after exit.

When you use the PMON utility the VxD stays loaded by default. This enables the VxD to keep previous activation records and report them back later to the DLL. This way you can activate the tool from one application and stop it from another application and get the original activation event and data.

The VxD is non intrusive. It is small, 7KB and is not active if you don't call it. No interrupt chaining is used.

5.3 The DLL

The DLL implements a set of APIs and communicates with the VxD. It should be loaded by the application (using the PMON utility, or by loading it within the application code), and it is unloaded automatically when the last application that uses it exits.

5.4 The Utility Main features

5.4.1 Event Selection

You can select an event using the index drop-down list box or using the event drop-down list box. Choosing the first letter of the string will move the selection to the first event that starts with that letter. This will speed up the event selection.

When you exit, the last settings are saved in the pmon.ini file in the same directory as the Pmon Utility. These events are selected as the defaults when the Pmon Utility starts again, if the counter are not running. If counters are running at launch time, the Pmon Utility will display the current counter names and values and these will be written to the notification window.

The variables used in event selection are (refer to Figure 3):

- **Event**: The name of the event being monitored.
- **Index:** The index of the event as used by the Pmon Utility.
- **Umask**: This drop-down list box exists only if the processor is Pentium Pro. The value here combines with the event value to select the actual event to count.
- **Ring**: The privilege level of the events to be watched.

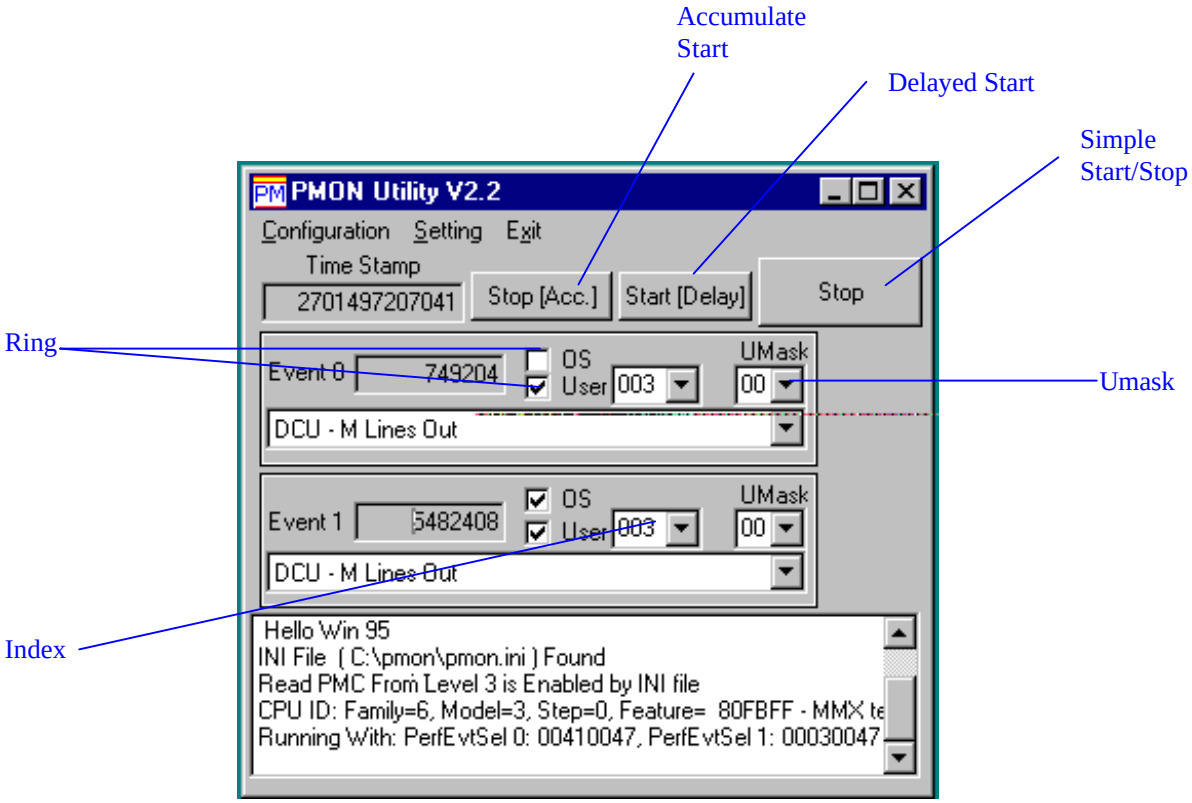

Figure 3

The files associated with event selection are:

- **PMON.INI**: Stores the last selected events and initializes the next instance of the Pmon Utility to these events.
- **EVENTS.INI**: Modify this to add events not currently programmed into the Pmon Utility.

5.4.2 Start/Stop

Refer to Figure 3 for the following:

- **Simple Start/Stop**: Use this for immediate starting and stopping of the counters.

- **Start [Delay]:** For a delayed start of the counters, first click on "Configuration" from the menu bar and then set the following parameters in the Configuration dialog box (see Figure 4):
    1. Delay after Start: Select the number of seconds to wait before starting the counters.
    2. Auto Stop After: You can select when you want to stop counting by entering the number of seconds here, or you can click on the simple Stop button to stop the counters immediately.

- **Start [Accumulate]:** A Windows timer is set to the requested interval and on each tick, it logs the counters values to a table in memory. The accuracy of this timer is at the mercy of the Windows OS and the running applications at that time. This might be okay as we are logging the cycles elapsed and most of the time we are interested in the difference between consecutive entries in the table. The results are written a file named trace.txt in the Pmon directory.

To use the Accumulate Start, first click on “Configuration” from the menu bar and then set the following parameters in the Configuration dialog box (see Figure 4):

1. Number of Samples: Select the number of samples to take.
2. Sampling Interval: Select the length of the interval between each sample (in milliseconds).
3. Message window (optional): You can enter a descriptive line here to be written to trace.txt.

- **Stop message at the notification window:** Refer to Figure 3. When counters are stopped the Pmon utility displays a summary of the last run in the notification window. The information provided is:
  - The actual programmed hexadecimal values of the performance registers
  - The event string and the event index that were chosen for counter 0 and counter 1
  - The ring selection
  - The counter values and the elapsed cycles from start to stop

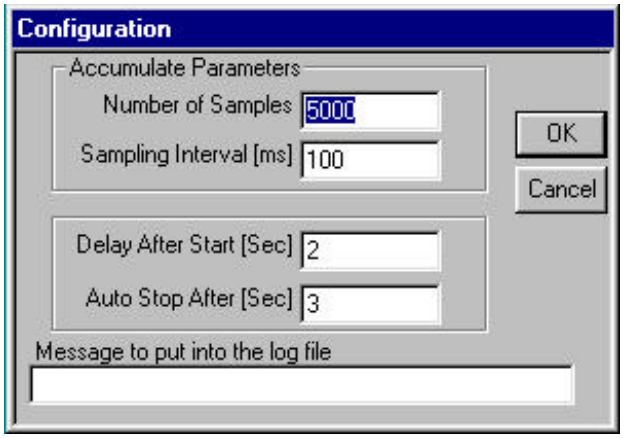

Figure 4

5.4.3 Additional Settings

These settings are accessed by selecting the “Setting” option on the menu bar (see Figure 5).

- **Auto Stop**: Check this option to have the counters always auto stop after the time specified in the Configuration dialog box (see Figure 4).

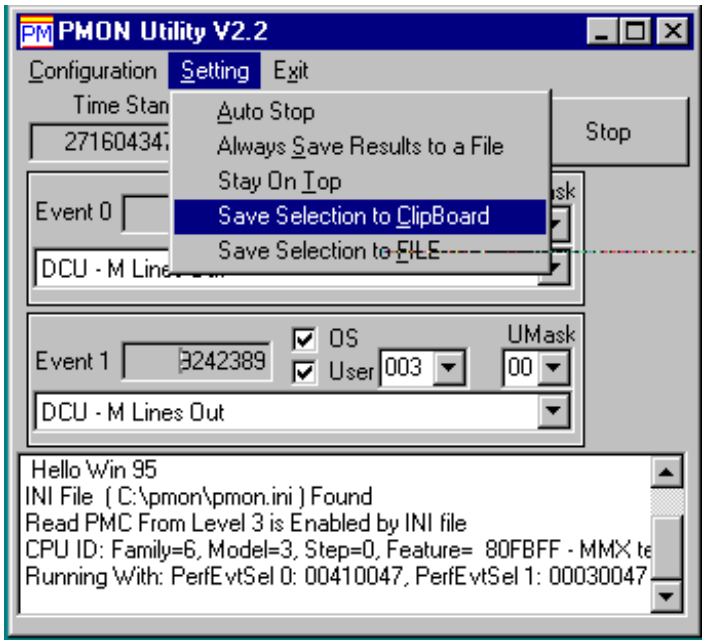

Figure 5

- **Always save results to file**: Check this option to always have the counter data saved to a log file. The file will be named pmon32.log and it will be saved in the same directory as the Pmon Utility. If the file already exists, the new data will be appended to the end of the file.

- **Stay on top**: Check here to keep the Pmon Utility always on top.

- **Save Selection to Clipboard**: Highlight an area in the Notification window and then click here to have the area copied to the Windows clipboard.

- **Save Selection to FILE:** Highlight an area in the Notification window and then click here to have the area copied to a file. The file name will be named trace.txt and it will be saved in the same directory as the Pmon Utility. If the file already exists, the new data will be appended to the end of the file.

5.5 Command Line Operation

5.5.1 Command Line Parameters

Table 2 lists the parameters that can be used with Pmon in the command line mode of operation.

Table 2: The Utility Command Line options

Command	Description	Example
/start	Start the counters now	/start
/T0=n /T1=m	Specify the index for the event you want to activate. See Appendix A for the events and their indices.	/T0=5 /T1=20
/T0CPL=x /T1CPL=y	Set the ring level at which you want to count the events. 30 - level 012 and 3 3 - only level 3 0 - only level 0	/T0CPL=30 - counter 0 counts all relevant events if code runs in ring 0,1,2,3 /T1CPL=3 - counter 1 counts all relevant events if code runs in ring 3
/batch	Activate the utility without showing the GUI/interactive mode and exits at the end. The VxD will stay loaded after exit.	/BATCH
/MSG=zzzyyyxxx	Write this string to the log file as a reference.	/MSG=Test20_Just_Before_Lunch
/STOP	Stop the counter, read results and report	/stop
/FILE=xyz.log	When stopped, will log the results to this file (appended to the end) with regular date and time stamp. Note, no spaces around the equal sign	/file=July23.log
/ACC_START	Start counting and sample the counter every period of time (/INTERVAL). Put readings into a buffer. At the end write the whole table into a file-TRACE.TXT	/ACC_START /SAMPLS=5000 INTERVAL=1000
/SAMPLES=	Number of samples	see above example
/INTERVAL=	Sample rate [in ms]	see above example
/DELAY_START	Start counting after the delay.	/DELAY_START /DELAY=5
/DELAY=	Delay [sec]	see above example

Note: The command line is not case sensitive.
Table 3 summarizes the parameters that are available with each mode of operation.

Table 3: Main Command line options Summary

Purpose	**/BATCH**	**/ACC_START /INTERVAL= /SAMPLES=**	**/START**	**/STOP**	**/DELAY_START /DELAY=**	**/FILE= /MSG=**
Start & Exit	X		X			
Stop & Exit	X			X		X
Start with Delay & Exit	X				X	
Accumulate & Exit	X	X				

5.5.2 Command Line Examples

The following are examples of command lines.

```
1.    pmon  /t0=1 /T1=12 /t0cPl=30 /T1CPL=0 /StaRt /batch
```

Timer 0 is selected with event #1 with level 0,1,2,3
Timer1 is selected with event #12 with level 0,1,2,
Start the timers now.
Don't show the pmon GUI, just exit at the end.

```
2.    pmon32 /stop /file=log.log  /msg= Test20_Just_Before_Lunch  /batch
```

These are the results in the log.log file:

```
--------- Log xxx---------
Fri Sep 20 13:04:00 1996
CPU ID: Family= 5, Model= 2, Stepping= 11, FeatureFlags=      1BF
User Message: Test20_Just_Before_Lunch
Counter 0:  1294383
Counter 1:  11030874
Elapsed cycles (TSC):   11300
Cnt0 E:     02570243    Ring: 0          Mask0=00
Cnt1 E:     00000000    Ring: 0          Mask1=00
Event 0[4]: Data Read Miss
Event 1[23]:Instructions Executed in the v-pipe e.g. parallelism
--------- Log End------
```

6. PMON Data Files

6.1 INI files / Log files:

6.1.1 PMON.INI

The pmon.ini file contains start up configuration lines, which may be edited as needed. An example is shown in Figure 6. If you are using the Pmon Utility GUI, pmon.ini will be updated automatically each time you close the application.

PMON.INI

```
[Initialization Values]
Read PMC From Level 3=ON

Event 0 Last Selection=27
Event 1 Last Selection=2

Event 0 Last Ring Selection=0
Event 1 Last Ring Selection=0

Event 0 Last Mask Selection=0x00
Event 1 Last Mask Selection=0x00

Interval [in ms]=1000
Delay [in ms]=4
Buffer Size=5
Auto Stop Time [in sec]=3
```

Figure 6

Parameter Explanation:

Read PMC From Level 3

The options are ON or OFF. Default is ON. This instruction is only available on Pentium Processor with MMX technology and on Pentium Pro family. It allows reading the counters directly from ring 3.

All the other parameters

Used by the utility to save last usage before exit and restore those entries when launched again. Note that if the counters are running on the next launch, these parameters are not used and the current counter setting are used.

6.1.2 EVENTS.INI

The events.ini file contains information about additional events which are not hard-coded into the Pmon Utility. An example is shown in Figure 7 below. A description of how to add events to Pmon through the events.ini file follows.

EVENTS.INI

```
 [Pentium Additional Events - Counter 0]
Number Of New Events=0

[Pentium Additional Events - Counter 1]
Number Of New Events=0

[Pentium Pro Additional Events]
Number Of New Events=27
....
E_str_11=Transitions FP  to MMX
E_enc_11=0xCC
E_msk_11=0x01
E_idx_11=92

E_str_12=My favorite Event
E_enc_12=0x17
E_msk_12=0x01
E_idx_12=0
...
```

Figure 7

Parameter Explanation:

Note: The parameters are case sensitive!!!

There are 3 basic sections:

[Pentium Additional Events - Counter 0]
[Pentium Additional Events - Counter 1]
[Pentium Pro Additional Events]
The utility detects under which processor it runs and looks for the appropriate section.

Number Of New Events=27

This parameter specifies how many new events exist in this section. In this example 27 new events should follow this section. Each event should have four lines with a consecutive index (starting from 1) embedded into the parameter name. For example E_str_12, refers to the 12th entry.

E_str_n=

a string that represents this event. Will be used in the drop-down list box and for the data files.

E_enc_n=

a hex number (example: 0x36) which is the event's encoding.

E_msk_n=

a hex number (example: 0xFF) which is the event's Umask (relevant for Pentium Pro Processor Family).

E_idx_n=

A decimal number (starting from 0) for the index in the Application/DLL event data base. It inserts the new event into this entry of the data base and overrides the previous event.

The first unoccupied entries in the data base for this version of the utility are:
Pentium processor: 60
Pentium Pro Processor: 82
The table contains 120 entries.

6.1.3 Log File

Figure 8 shows an example of the log file that is generated when you ask for this option, either by setting "Always Save Results to a File" from the "Setting" menu, or by command line parameter.

Log File

```
--------- Log xxx---------
Fri Sep 20 13:04:00 1996
CPU ID: Family= 5, Model= 2, Stepping= 11, FeatureFlags=      1BF
User Message: Test2A
Counter 0:   1294383
Counter 1:   11030874
Elapsed cycles (TSC):     11300
Cnt0 E:      02570243     Ring:  0     Mask0=00
Cnt1 E:      00000000     Ring:  0     Mask1=00
Event 0[4]:  Data Read Miss
Event 1[23]: Instructions Executed in the v-pipe e.g. parallelism
--------- Log End------
```

Figure 8

The file is created if it does not exist and then appended with this information. It includes the time the data was written, along with information about the CPU. Also included are the user message (if given) and the data from the counters.

6.1.4 Trace File

Figure 9 shows an example of the trace file that is generated when you select the accumulate mode, either by clicking on the "Start[Acc.]" button, or by command line parameter.

Trace.txt

```
--->> Trace From: Fri Sep 20 14:08:21 1996

CPU ID:      Family Model  Stepping     FeatureFlags
                5     2          11            1BF
User Message: My second attempt
Interval used:      500    [ms]
Cnt0 Ring:   0      Mask0=00
Cnt1 Ring:   0      Mask1=00
Event 0[32]: Hardware Interrupts
Event 1[27]: FLOPs

Sample#      Cnt0   Cnt1   Elapsed Cycles
00001        34     25     46157524
00002        14     44     19760606
00003        35     24     46143440
00004        14     43     19775050
00005        33     24     46153248
00006        16     43     19757254
00007        37     24     46153758
00008        14     43     19758794
00009        33     24     46143502
00010        16     43     19765716
00011        32     22     41175440
00012        15     45     19746536
00013        34     23     46160212
00014        17     44     19761174
00015        35     23     46176638
00016        15     44     19731874
00017        36     23     46156850
00018        15     44     19752132
00019        40     23     46158564

Total:       485    628    634388312

--------- End ------
```

Figure 9

Each entry here represents the incremental counts that occurred between two consecutive samplings. For example, sample # 2 reads “14 counter 0”. This means that Counter 0 was incremented by 14 after sample #1 was taken.
Note: The above format is a tab separated in the file, which can be easily imported into a spread sheet (CSV format).

7. The PMON APIs:

The APIs defined in PM32_DLL.DLL are listed in Table 4 below.

Table 4: The PMON APIs

Category	API	Description
	Pmon32Init	Load/connect to the VxD.
Init	Pmon32InitEx	Load/connect to the VxD + Option to unload VxD on exit
	Pmon32Close	Close connection to DLL/VxD. Depending on how the DLL was initialized, this may or may not unload the VxD.
	Pmon32InstallEvent	Install a new event into a requested entry in the DLL event data base. This overrides any existing entry. The table below shows the occupied events in the DLL data base. Pentium Pro Processor Events occupy entries from 0-81. Pentium Processor Events occupy entries from 0 - 61. It is advisable to add your events to the unoccupied entries. Both tables have 120 entries.
	Pmon32DisableRDPMCRing3 Pmon32EnableRDPMCRing3	Allows/disables reading the counters from ring3. Only available on Pentium Processors with MMX Technology or Pentium Pro Processors.
Operation	Pmon32Start	Program the events, start counting
	Pmon32StartDirect	Send the value directly to the counter control registers. This also attaches strings to these events so that when these counters are stopped and the EventToStr routines are called, these strings will be returned.
	Pmon32StartEx	Program the events, start counting. Additional support for Umask field.
	Pmon32StopAndReadCounters	Stop counters, read results.
Status	Pmon32ReadCounters	Read counters on the fly.
	Pmon32ReadTSC	Read the time stamp counter.
	Pmon32Status	Running/stopped, which events were or are being counted.
	Pmon32AreCountersRunning	TRUE/FALSE for counters running.
	Pmon32Event0ToStr Pmon32Event1ToStr	Returns a pointer to string for a given event index.

The formal definitions for these APIs are (as included in the file DLL_IF.H):

```
DWORD  (FAR *Pmon32Init)               (struct      Pmon32Version *ver);
DWORD  (FAR *Pmon32Close)              (void);
DWORD  (FAR *Pmon32Status)             (struct      Pmon32Reply   *Reply);
BOOL   (FAR *Pmon32AreCountersRunning) ();
```

```
DWORD  (FAR *Pmon32Start)               (BYTE Event0, BYTE Event0Ring,
                                         BYTE Event1, BYTE Event1Ring);
DWORD  (FAR *Pmon32StopAndReadCounters) (struct      Pmon32Reply  *Reply);
DWORD  (FAR *Pmon32ReadCounters)        (struct      Pmon32Reply  *Reply);
DWORD  (FAR *Pmon32ReadTSC)             (struct      Pmon32Reply  *Reply);
DWORD  (FAR *Pmon32EnableRDPMCRing3)    (void);
DWORD  (FAR *Pmon32DisableRDPMCRing3)   (void);
char   *(FAR *Pmon32Event0ToStr)        (BYTE        index);
char   *(FAR *Pmon32Event1ToStr)        (BYTE        index);

DWORD  (FAR  *Pmon32StartEx)            (struct      Pmon32StartCmd     *Cmd);

DWORD  (FAR  *Pmon32StartDirect)        (DWORD E0,   DWORD E1, char *str0,
                                        char *str1);

DWORD  (FAR  *Pmon32InstallEvent)       (BYTE Cnt, BYTE Index,
                                        BYTE  Encoding, BYTE Mask, char *str);

DWORD  (FAR  *Pmon32InitEx)             (struct      Pmon32StartCmd     *Cmd,
                                        struct Pmon32Version *ver);
```

The application communicates with the DLL using the following structure:

```
struct Pmon32StartCmd{
       BYTE   Event0;      // Index as defined by the table below (Section 9)
       BYTE   Event1;      // Index as defined by the table below (Section 9)
       BYTE   Event0Ring;  // 0 - ring 0, 3 - ring 3, 30 - ring 3 and 0
       BYTE   Event1Ring;

       BYTE   Event0Mask;  // Only for Pentium Pro processors. As defined by
the                        // architecture
       BYTE   Event1Mask;
       BYTE   Flg1;        // res
       BYTE   Flg2;        // res

       BOOL   unload_vxd_on_exit;       // TRUE - VxD will be unloaded on exit
       BOOL   Flg3;
       BOOL   Flg4;
       BOOL   Flg5;

       DWORD  Res1;
       DWORD  Res2;
       DWORD  Res3;
};

struct Pmon32Reply{
       DWORD  T0_l;        // Counter/event 0 LSB
       DWORD  T0_h;        // Counter/event 0 LSB
       DWORD  T1_l;        // Counter/event 1 MSB
       DWORD  T1_h;        // Counter/event 1 MSB
       DWORD  TSC_l;             // current Time Stamp LSB
       DWORD  TSC_h;             // current Time Stamp MSB
       DWORD  ElapsedTime_l;
       DWORD  ElapsedTime_h;
       DWORD  TSCLast_l;
       DWORD  TSCLast_h;
       BYTE   Event0;      // Index as known by the DLL
       BYTE   Event1;
       BYTE   Event0Ring;
       BYTE   Event1Ring;
       BYTE   Event0Mask;
       BYTE   Event1Mask;
       BYTE   pad1  ;            // structure alignment
       BYTE   pad2  ;
       DWORD  status;
```

```
        DWORD  res1;
        DWORD  res2;
        DWORD  res3;
        DWORD  res4;
        DWORD  res5;
        DWORD  res6;
};

struct Pmon32Version{
        char   VxDVersion[32];     // A string
        char   DLLVersion[32];     // A string
        int    VxDMajor;           // Represents the VxD major version number
        int    VxDMinor;           // Represents the VxD minor version number
        int    DLLMajor;           // Represents the DLL major version number
        int    DLLMinor;           // Represents the DLL minor version number
        char   res2[16];
};
```

8. Considerations

1. This version runs only on Windows* 95.

2. One can use batch mode to set the required events and set the option of reading the counters from level 3 (This is the default if this line is omitted). Then, the application can read the results (the counters) directly from its code (ring 3).

3. Two or more instances of the PMON Utility are allowed. Notice that the second instance will refresh its event selection only once at start, but won't be able to report on the right selection dynamically as the changes take place by other instances. On the other hand, it will show the changes in the counter totals. Also, if the second instance is the one that initiates the stop, it will report the right event.

4. The Pmon utility does not use all the options provided in the performance monitoring of the processor. You can observe the exact programming value that the tool is using by looking at the stop message in the notification window. However the APIs allow the programmer to directly interface with the register and utilize more features.

5. Notice that the Pentium Pro events are programmed with E bit set to 0. This means that all events are a duration type and not occurrence one.

6. In the Pentium Pro case some events are only applicable to a specific counter. The current version of the Pmon utility does not prevent you from using it with the wrong counter. However the description of the event notifies you with the correct counter to use. See the Pentium Pro event table, events 55-60.

9. The Events

The following are the events and their indices. These are the default programming in the application data base and in the DLL data base. It also correlates to the result indices in the log file.
Each table consumes 120 entries and the unoccupied ones are different from Pentium to Pentium Pro tables.
Note that indices below are the indices defined by the tool and they do NOT necessarily match the ones described in the processor programmer reference manual .

Pentium Processor Events for Counter 0

Index	Event	Encoding
0	Data Read	0x00
1	Data Write	0x01
2	Data Read or Data Write	0x28
3	Data TLB Miss	0x02
4	Data Read Miss	0x03
5	Data Write Miss	0x04
6	Data Read or Data Write Miss	0x29
7	Write (hit) to M or E state lines	0x05
8	Data Cache Lines Written Back	0x06
9	External Snoops	0x07
10	External Data Cache Snoop Hits	0x08
11	Memory Accesses in Both Pipes	0x09
12	Bank Conflicts	0x0a
13	Misaligned Data Memory or I/O References	0x0b
14	Code Read	0x0c
15	Code TLB Miss	0x0d
16	Code Cache Miss	0x0e
17	Any Segment Register Loaded	0x0f
18	Branches	0x12
19	BTB Hits	0x13
20	Taken Branch or BTB Hit	0x14
21	Pipeline Flushes	0x15
22	Instructions Executed	0x16
23	Instructions Executed in the v-pipe e.g. parallelism	0x17
24	Locked Bus Cycle	0x1c
25	I/O Read or Write Cycle	0x1d
26	Non-cacheable memory reads	0x1e
27	FLOPs	0x22
28	Breakpoint match on DR0 Register	0x23
29	Breakpoint match on DR1 Register	0x24
30	Breakpoint match on DR2 Register	0x25
31	Breakpoint match on DR3 Register	0x26
32	Hardware Interrupts	0x27
33	Clocks while a bus cycle is in progress (bus util.)	0x18
34	Number of clocks stalled due to full write buffers	0x19
35	Pipeline stalled waiting for data memory read	0x1a
36	Stall on write to an E or M state line	0x1b
37	Pipeline stalled due to addr generation interlock	0x1f
38	Bus ownership latency	0x2a
39	MMX instructions executed in U-pipe	0x2b
40	Number of L1 M-state line sharing	0x2c
41	EMMS instructions executed	0x2d
42	Bus utilization due to processor activity	0x2e
43	Saturated MMX instructions executed	0x2f
44	Cycles not in HLT state	0x30

Pentium Processor Events for Counter 0 (continued)

Index	Event	Encoding
45	MMX data memory reads	0x31
46	Floating point stalls	0x32
47	D1 starving but instruction FIFO empty	0x33
48	MMX data memory writes	0x34
49	Pipeline flushes due to wrong branch prediction	0x35
50	MMX misaligned data memory reference	0x36
51	Returns miss predictions	0x37
52	MMX Multiply interlock	0x38
53	Returns executed	0x39
54	BTB bogus entry detected	0x3a
55	MMX writes backed - pipe stalled	0x3b
56	Reserved	0x3c
57	Reserved	0x3d
58	Reserved	0x3e
59	NULL event	0x3f
60	Reserved	0xFF
61	Reserved	0xFF
....		
118	Reserved	0xFF
119	Reserved	0xFF

Pentium Processor Events for Counter 1

Index	Event	Encoding
0	Data Read	0x00
1	Data Write	0x01
2	Data Read or Data Write	0x28
3	Data TLB Miss	0x02
4	Data Read Miss	0x03
5	Data Write Miss	0x04
6	Data Read or Data Write Miss	0x29
7	Write (hit) to M or E state lines	0x05
8	Data Cache Lines Written Back	0x06
9	External Snoops	0x07
10	External Data Cache Snoop Hits	0x08
11	Memory Accesses in Both Pipes	0x09
12	Bank Conflicts	0x0a
13	Misaligned Data Memory or I/O References	0x0b
14	Code Read	0x0c
15	Code TLB Miss	0x0d
16	Code Cache Miss	0x0e
17	Any Segment Register Loaded	0x0f
18	Branches	0x12
19	BTB Hits	0x13
20	Taken Branch or BTB Hit	0x14
21	Pipeline Flushes	0x15
22	Instructions Executed	0x16
23	Instructions Executed in the v-pipe e.g. parallelism	0x17
24	Locked Bus Cycle	0x1c
25	I/O Read or Write Cycle	0x1d
26	Non-cacheable memory reads	0x1e
27	FLOPs	0x22
28	Breakpoint match on DR0 Register	0x23

Pentium Processor Events for Counter 1 (continued)

Index	Event	Encoding
29	Breakpoint match on DR1 Register	0x24
30	Breakpoint match on DR2 Register	0x25
31	Breakpoint match on DR3 Register	0x26
32	Hardware Interrupts	0x27
33	Clocks while a bus cycle is in progress (bus util.)	0x18
34	Number of clocks stalled due to full write buffers	0x19
35	Pipeline stalled waiting for data memory read	0x1a
36	Stall on write to an E or M state line	0x1b
37	Pipeline stalled due to addr generation interlock	0x1f
38	Bus ownership transfers	0x2a
39	MMX instructions executed in V-pipe	0x2b
40	Number of L1-line sharing	0x2c
41	MMX ---> FP transitions	0x2d
42	Writes to non-cacheable memory	0x2e
43	Saturated MMX operations executed	0x2f
44	Ex stage stalled due to a D-TLB miss	0x30
45	MMX data read misses	0x31
46	Taken branches	0x32
47	D1 starving but only 1 inst in inst FIFO	0x33
48	MMX data write misses	0x34
49	Pipeline flushes-wrong branch prediction in WB-Stage	0x35
50	MMX wait memory read - pipe stalled	0x36
51	Returns predicted (correctly & uncorrectly)	0x37
52	MMX store stall due to previous operation	0x38
53	RSB overflows	0x39
54	BTB miss-prediction or a not-taken branch	0x3a
55	Stall on MMX write to E or M line	0x3b
56	Reserved	0x3c
57	Reserved	0x3d
...		
118	Reserved	0xFF
119	Reserved	0xFF

Pentium Pro Processor Events

Index	Encoding	Umask	Event
0	0x43	0x00	DCU - Data Mem Refs
1	0x45	0x00	DCU - Lines In
2	0x46	0x00	DCU - M Lines In
3	0x47	0x00	DCU - M Lines Out
4	0x48	0x00	DCU - Miss Outstanding
5	0x80	0x00	IFU - iFetch
6	0x81	0x00	IFU - iFetch Miss
7	0x85	0x00	IFU - ITLB Miss
8	0x86	0x00	IFU - Mem Stall
9	0x87	0x00	IFU - Inst Lrngth Decoder Stall
			L2 Cache
10	0x28	0x0F	L2 Instructions Fetches [MESI]
11	0x29	0x0F	L2 Data Loads [MESI]
12	0x2A	0x0F	L2 Data Stores [MESI]
13	0x24	0x00	Lines Allocated in L2
14	0x26	0x00	Lines Removed from L2
15	0x25	0x00	Modified Lines ALLOCATED in L2
16	0x27	0x00	Modified Lines REMOVED from L2

Pentium Pro Processor Events (continued)

Index	Encoding	Umask	Event
17	0x2E	0x0F	L2 Requests [MESI]
18	0x21	0x00	L2 Address Strobes
19	0x22	0x00	Cycles Data Bus was BUSY
20	0x23	0x00	Cycles Data Bus was BUSY in READ
			External Bus Logic (EBL)2
21	0x62	0x00	DRDY is asserted [Self/Processor]
22	0x62	0x20	DRDY is asserted [Any/any agent]
23	0x63	0x00	LOCK is asserted [Self]
24	0x63	0x20	LOCK is asserted [Any]
25	0x60	0x00	BUS requests outstanding
26	0x65	0x00	Burst Read transactions [Self]
27	0x65	0x20	Burst Read transactions [Any]
28	0x66	0x00	Read for Ownership Transactions [Self]
29	0x66	0x20	Read for Ownership Transactions [Any]
30	0x67	0x00	Write Back Transactions [Self]
31	0x67	0x20	Write Back Transactions [Any]
32	0x68	0x00	Instruction FETCH Transactions [Self]
33	0x68	0x20	Instruction FETCH Transactions [Any]
34	0x69	0x00	BUS Invalidate Transactions [Self]
35	0x69	0x20	BUS Invalidate Transactions [Any]
36	0x6A	0x00	BUS Partial Writes [Self]
37	0x6A	0x20	BUS Partial Writes [Any]
38	0x6B	0x00	BUS Partial Transactions [Self]
39	0x6B	0x20	BUS Partial Transactions [Any]
40	0x6C	0x00	BUS Trans IO [Self]
41	0x6C	0x20	BUS Trans IO [Any]
42	0x6D	0x00	BUS DEFFERED Transactions [Self]
43	0x6D	0x20	BUS DEFFERED Transactions [Any]
44	0x6E	0x00	BUS BURST Transactions [Self]
45	0x6E	0x20	BUS BURST Transactions [Any]
46	0x70	0x00	BUS- ALL transactions [Self]
47	0x70	0x20	BUS- ALL transactions [Any]
48	0x6F	0x00	BUS- Memory Transactions [Self]
49	0x6F	0x20	BUS- Memory Transactions [Any]
50	0x64	0x00	BUS- cycles processor RECEIVING DATA
51	0x61	0x00	BUS- Cycles processor driving BNR pin
52	0x7A	0x00	BUS- Cycles processor driving HIT pin
53	0x7B	0x00	BUS- Cycles processor driving HITM pin
54	0x7E	0x00	BUS is SNOOP Stalled
			Floating Point Unit
55	0xC1	0x00	FP Ops RETIRED [Counter 0 Only]
56	0x10	0x00	FP Ops EXECUTED [Counter 0 Only]
57	0x11	0x00	FP Assist [Counter 1 Only]
58	0x12	0x00	FP MUL [Counter 1 Only]
59	0x13	0x00	FP DIV [Counter 1 Only]
60	0x14	0x00	FP Cycles Divider is Busy [Counter 0 Only]
			Memory Ordering
61	0x03	0x00	Mem Ord - LD Blocks
62	0x04	0x00	Mem Ord - SB Drains
63	0x05	0x00	Mem Ord - Misalign Mem Refs
			Instruction Decoding and Retirement
64	0xC0	0x00	INST - Instructions Retired
65	0xC2	0x00	INST - uOps Retired
66	0xD0	0x00	INST - Instructions decoded
			Interrupts
67	0xC8	0x00	INT - Hardware INTERRUPTS received
68	0xC6	0x00	INT - Cycles INterrupts are DISABLED

Pentium Pro Processor Events (continued)

Index	Encoding	Umask	Event
69	0xC7	0x00	INT - Cycles Ints are Pending But disabled
			Branches
70	0xC4	0x00	Branch Insts Retired
71	0xC5	0x00	BR Miss PRED Retired
72	0xC9	0x00	BR Taken Retired
73	0xCA	0x00	BR Miss PRED Taken Retired
74	0xE0	0x00	BR Insts Decoded
75	0xE2	0x00	BRs That miss the BTB
76	0xE4	0x00	BR Bogus
77	0xE6	0x00	BACLEAR is asserted
			Stalls
78	0xA2	0x00	STALLS - Resource Stalls
79	0xD2	0x00	STALLS - Partial RAT Stalls
			Segment Register Loads
80	0x06	0x00	Segment Reg Loads
			Clocks
81	0x79	0x00	CPU CLK UnHalted
82	0xFF	0xFF	Reserved
83	0xFF	0xFF	Reserved
84	0xFF	0xFF	Reserved
85	0xFF	0xFF	Reserved
86	0xFF	0xFF	Reserved
87	0xFF	0xFF	Reserved
88	0xFF	0xFF	Reserved
89	0xFF	0xFF	Reserved
90	0xFF	0xFF	Reserved
91	0xFF	0xFF	Reserved
92	0xFF	0xFF	Reserved
...			
118	0xFF	0xFF	Reserved
119	0xFF	0xFF	Reserved