TurionPowerControl Software documentation

Version 0.41 - 2011/09

General Index

1. Introduction

TurionPowerControl born as a software to control Turion ZM/RM/QL processor parameters. Currently it supports Family 10h, 11h, 12h and 14h processors.

TurionPowerControl allows to tweak processors parameters such as VID, DID and FID, C1E registers, Hardware Thermal Control (HTC) features, and many others options. Also gives the features to monitor preocessor pstate changes, temperatures and precise cpu load. Those parameters are useful in a large manner to tweak and optimize your Griffin/Lion processor the way you want.

- **Family 10h** is made up of processors such as Phenom, Phenom II, Athlon II (both mobile M300 series and desktop series) and Turion II processors (mobile M500/M600 series, Pxxx series and Nxxx series)
- **Family 11h** is made up of processors such as Turion ZM/RM and Athlon QL.
- **Family 12h** is related to the Llano platform and is made up of A-series and E2-series processors
- **Family 14h** is related to the Brazos platform and is made up of Zacate and Ontario processors (AMD E-series and C-series).

This is software is provided AS IS, with no guarantees at all, and I will not be responsible for any damage you may cause to your computer using this program.

1.1 I have an Athlon 64, what about me?

Athlon 64 (and all processors belonging to the same Family, like Turion 64, Athlon 64 X2, older Semprons, etc... etc...) can't use this program.

Those older processors have a very very different approach to power management, since all the transition is done in software. They don't have a pstates table to manipulate, because they require a software driver to do power management.

For this reason, adapting this program is really too hard and requires a complete rewrite. Also such a thing would require deep access to some operating system structures.

Unfortunately this program will never support such processors, but you should not desperate because:

- if you are a Windows user, you can try RMClock program that will most probably work fine with your processor
- if you are a Linux user, you may take a look to **linux-phc** (http://www.linux-phc.org/). It is a substitute driver for powernow-k8 that deals about power management on modern linux distributions, and will allow you to downclock and downvolt your Athlon 64 processor.

1.2 Requirements for Windows

The program will run fine on

- Windows XP (32-bit and 64-bit)
- Windows Vista (32-bit and 64-bit)
- Windows 7 (32-bit and 64-bit)

You may need to install **Microsoft Visual C++ 2008 runtimes**, expecially if you are running on an older operating system.

If you have a 32 bit operating system, you can download the latest runtimes from here

If you have a 64 bit operating system, you can download the latest runtime from here

Also mind that you need the **administrator privileges** to let the program run!

1.3 Requirements for Linux

The program has been reported to compile and run fine with a number of different distributions (Fedora, OpenSuse, Ubuntu, Slackware, ...).

You just need a recent gcc suite compiler (version >= 4.4) and the **cpuid** and **msr** modules to be loaded in your system.

If you are unsure about cpuid and msr modules, you can run the command:

```
> find /dev/cpu
```

and you should get something like this:

```
/dev/cpu
/dev/cpu/microcode
/dev/cpu/1
/dev/cpu/1/cpuid
/dev/cpu/1/msr
/dev/cpu/0
/dev/cpu/0/cpuid
/dev/cpu/0/msr
```

in which every processor/core has its own cpuid and msr file (highlighted in **bold**). Microcode file is not necessary at all.

If you don't obtain such a result, you may want to run **modprobe** (as root!) to let the modules get loaded:

```
> modprobe cpuid
```

```
> modprobe msr
```

If you have troubles getting the modules loaded (modprobe says it can't find the modules, for example) you can try to download the modules for your distribution from your favourite package manager and then try to modprobe them again.

You may also need to check that the kernel exposed the PCI bus configuration registers with this command:

```
> find /proc/bus/pci/00/18.3
```

you should get this to be safe:

```
/proc/bus/pci/00/18.3
```

If you get a file not found message, you probably need to upgrade your kernel or recompile it. Usually support for userland access to pci registers is builtin into recent kernels.

Anyway, be always sure you run the program as **root** user, else you won't get access to the processor parameters.

A last note: many people, after using the program to change frequency, then goes into **/proc/cpuinfo** to see if it had any effect. This is not a good idea: /proc/cpuinfo is not updated calling the real hardware! The most effective way is to do a benchmark. I can reccomend prime95, which has an integrated benchmark that shows if your processor is running at a slower or faster frequency, but any benchmark is suitable for the task.

The same applies to other software tools like **cpufreq** utilities: they usually claim using hardware calls, but actually this is not entirely true, and so they give not updated frequency and voltage results.

1.4 Changelog

Version	Changes
0.41	- Polished and updated documentation with new options and behaviours. Now almost all the switches depends on nodes activated with **-node** switch and cores activated with **-core** switch. Check the documentation again because some commands have been changed. - Initial support for Brazos platform (i.e. Zacate and Ontario processors, Family 14h) and Llano platform (i.e. A-series and E2-series processors, Family 12h)

	- Corrected some DRAM reports for Family 10h processors. - Scaler has been reactived in a very experimental fashion. Has not been tested at all on processors with more than two cores. - Introduced a new interface for performance counters. - Changed cpu usage performance counter switch from **-cpuusage** to **-perf-cpuusage**. Introduced also **-perf-fpuusage** and **-perf-dcma** counters.
0.40a	- Underlying program structure has been completely revamped to fully support multiprocessor and multinode machines. Check the documentation for new and removed commands - **pf, pd, pv, pall, pallc** commands have been removed. FID, DID and VID manipulation are now managed via **-set** switch. - Full DRAM timings reporting - Extended hypertransport status reporting (link coherency, link frequency and width, etc...) - Scaler has been disabled in this version, since it requires major modifications. It is a low priority and will be reactivated as soon as possible. - Compilation has changed, now it accomplished via **make** command. Check Chapter 5 for easy instructions. - Project is now hosted on **Google Code** servers at the address http://code.google.com/p/turionpowercontrol/ . You may find here latest development code here.
0.31	- Fixed maximum northbridge frequency reporting - Added a -forcenumcores switch to force the number of cores. This is proved to be useful to let some TurionPowerControl features (frequency and voltage control, mainly) work with multiprocessor machines on Unix/Linux platforms. Use it with caution, it may cause undefined behaviours. - Extended support for up to 64 cores machines (using -forcenumcores switch) - Added -forcePVI and -forceSVI switches to force, respectively, Parallel VID mode or force Serial VID mode instead of autodetecting. -forceSVI in particular could be useful if the pstates power table shows wrong/non-coherent/too low voltage values. These switches are available only for Family 10h processors. - Fixed a misunderstanding implementation of PVI mode on K10 processors: AM2 board with AM2+/AM3 processors showed incorrect

	voltage values. Fixed the documentation too by consequence.
0.30	- Corrected an issue with minimum voltage VID with Parallel VID implementations. - Corrected an issue with **-set voltage** switch reporting wrong rounding when used on a specific core instead of all cores. The effect of the command was therefore successful. - Corrected a problem with cpu usage reporting with K10 processors with more than 2 cores. - Removed an annoying debug string when using **-set frequency** with K10 processors. - Added support for northbridge voltage and divisor change for K10 processors. Northbridge voltage can be changed with -nbvid switch or using -set switch. Divisor can be changed with -nbdid switch - Added support for northbridge voltage change in configuration file - Added some more usage examples, fixed some documentation issues
0.29.1 alpha	- Added a warning message at startup if program can't access CPUID or MSR registers that asks for administrative privileges and, for linux users, cpuid and msr modules. - Added the **-set** switch, which is really useful and much more user friendly than direct FID/DID/VID manipulation. Check the documentation and examples for usage. - Fixed detection of correct PVI/SVI mode for Family 10h processors - Updated documentation with all the program switches - Corrected some minor issues with K10 processors regarding altvidslamtime (unsupported on K10) and psithreshold switches - Added requirements paragraphs in documentation - Added the **frequency** and **voltage** items in the configuration file definition. They can be used instead of FID, DID and VID values for easier pstate manipulation. (See the included example.cfg file on how to use them). - Corrected a detection error with Turion II/Athlon II mobile processors. - Removed AltVIDSlamTime reporting for Family 10h processors (unsupported on them). - Added CPU Maximum Frequency reporting
0.29 pre-alpha	- Added preliminary support for Family 10h processor, like Phenom,

	Phenom II, Athlon II and Turion II processors. Only base functions are currently supported and has not been really tested enough. Please report any problem to my email, so I can fix bugs. Scaler is totally untested on Family 10h processors, so use it at your own risk! - Added some simple examples to this readme file
0.22	- Fixed Dual Plane/Triple plane operational mode reporting - Added some more interesting bit reporting (PSI_L enable/threshold, clock ramp hysteresis) - Added capability to control PSI_L bit enable/disable and threshold - Added C1E state reporting - Added C1E bit set enable/disable
0.21	- Fixed a MSR mask problem with -altvidslamtime switch.
0.20a	Alpha version. - Added CPU scaler daemon mode - Added external configuration file support - Added autorecall (autorefresh) daemon mode - Added performance counters information switches, including a very precise CPU Usage monitor daemon. - Added -CM switch to monitor CPU pstate switching. Useful to monitor pstate 6/7 abnormal transitions too.
0.12	- Added Hypertransport switches - Added HTC (Temperature control) switches to get current status and change settings.
0.10	First released version

2. Usage

The usage is simple, you can launch the program with no additional commands and it exposes all the available commands.

Available commands:

2.1 Node and core selecting switches

In version 0.40 and above has been introduced a distinction between **nodes** and **cores**.

A node in the system represents a physical processor. With core instead is intended exactly what it is commonly known, a core of a physical processor.

If you have a machine with a single processor (so just one node), you can safely discard the node definition and just deal with core selection.

These switches will **activate** the node(s) and the core(s) you will operate on, it means the nodes and the cores that will be affected by other switches that follow on the command line.

-node <nodeId>

This switch will select a specific physical processor in the system identified by an integer value <nodeId>, or all physical processors if <nodeId> is set to “all” (without quotes);

-core <coreId>

This switch will select a specific core inside the selected node in the system. A core is identified by an integer value <coreId>. If you want to select all cores, set <coreId> to “all” (without quotes)

By default, if you don't set explicitly any node and any core, the switches on the command line will automatically apply to all cores of all nodes.

2.2 Processor power management switches

These switches will let you manipulate the power management features of the processors in the system.

-l

Shows the table of pstates and some other informations. Each pstate has a vid, fid and did. FID and DID determine the actual frequency in that pstate with a given formula, while VID determines core voltage. (see below)

-spec

Shows some more processor family specific information. For example, will show C1E status on supported processors, or will show the northbridge frequency and voltage table, and so on...

-dram

Will show many details on DRAM timings collected from the integrated memory controller.

-en <pStateId>
Sets Enabled bit for pStateId on active node/core. Actually it does nothing except enabling a bit (eg: doesn't unlock any pstate, unfortunately)

-di <pStateId>
Inverse of -en

-fo <pStateId>

Forces active node/core to switch to (pStateId). You can force a transition over currently set maximum pstate if you first set maximum pstate with -psmax command.

-psmax <pStateId>
Sets maximum pstate Id for active node/core. Usually on Family 11h ZM and RM processors, this is set to default to value 2 (means that pstates to be used are 0, 1 and 2), while QL processors have this value set to 1 (means that pstates to be used are 0 and 1). This limit the number of lower pstates that processor can switch to. Unfortunately setting this parameter to anything above your processor default doesn't unlock automatic transition to any more pstate, but you can use -fo command switch to manually obtain a transition to specificied pstate.

-set <commands>
set switch is really useful for using a compact and understandable way to assign frequency and voltages to pstates and cores.
You can use some easy commands to assign frequency or voltage to a specific pstate to all cores in a single round or to specific core.

Commands are:

node <nodeIndex> **–** sets the current node you wish to work on. Notice that this is only useful if you have a multiprocessor machine. <nodeIndex> is an integer value or “all” to represent all nodes. By default all nodes are selected. Changing the node inside a -set switch won't affect the node activated in the main command line.

core <coreIndex> – sets the core you want to work on. Must be followed by and integer representing the core or the value “all” to mean that you want to tweak all cores in a single shot. By default all cores are selected. Changing the core inside a -set switch won't affect the core activated in the main command line.

pstate, **ps** <pstateIndex> – sets the pstate you want to work on. Must be followed by an integer representing the pstate you want to modify. By default pstate 0 is selected.

f, **freq**, **frequency** <frequency> – followed by an integer representing the frequency you want to set in Mhz.

vc, **vcore**, **voltage** <voltage> – followed by a decimal value representing the voltage you want to set.

nbvoltage, **nbvol**, **nbv** <voltage> **–** followed by a float representing the northbridge voltage you want to set.

fid <fid>**–** followed by a decimal or an integer representing the fid you want to set.

did <did> - followed by a decimal represnting the did you want to set.

vid <vid> – followed by an integer representing the vid you want to set.

Check the usage examples because this switch can really make you life easier, since you won't have to deal manually with FID, DID and VID values.

-nbvid <nbVid>

This command is available only for Family 11h processors. It sets northbridge (Integrated Memory Controller) operating voltage. The formula used is the same of processor VID.

-nbvid <pState> <nbVid>

This command is available only for Family 10h processors. It sets the northbridge (Integrated Memory Controller + L3 Cache) operating voltage to the selected pstate on active node. It automatically sets the desider nbVid to all cores The used formulas are the same of processor VID. **Note:** if you are going to use this switch, I suggest to use the more user friendly -set switch to set northbridge voltage.

-nbdid <pState> <nbDid>

This command is available only for Family 10h processors. It sets the northbridge divisor value for active node. NbDId can be 0 or 1, but it is not very useful since it can lock the processor to the slowest pstate.

2.3 Voltage control switches

-slamtime <register>

-altvidslamtime <register>

These command sets the time needed for voltage stabilization when a new voltage is programmed to the regulators. The higher the value, the more stable the system, but then a transition will require more time. Slamming is used only when processor is working with Serial VID Interface (so, mobile family 11h and dual plane family 10h processors will use voltage slamming). These commands applies on active node.

The switches require an integer value that is a **register**. These are the appropriate values:

0=10us

1=20us

2=30us

3=40us

4=60us

5=100us

6=200us

7=500us

-altvidslamtime is available only for Family 11h processors and manages the stabilization time between a transition from the slowest pstate VID to the AltVID voltage.

If you want to read more about, you can find the proper referements on the BDKG guides from AMD.

In particular, for Family 11h you can check Document 41256 AMD Family 11h BKDG Rev 3.00, register F3xD8, page 159.

For family 10h, you can check Document 31116 AMD Family 10h BKDG Rev. 3.06, register F3xD8, page 242

-rampuptime <register>

-rampdowntime <register>

Set the StepUpTime or StepDownTime for the processor. It is available only in Family 10h processors when they use the Parallel VID Interface (single plane mode). In this case the programmed voltage to the regulators is stepped to the destination, instead of being slammed as in the previous case. This commands applies on active node.

Values range from 0 to 15 and corresponds to:

0=400ns	1=300ns	2=200ns	3=100ns
4=90ns	5=80ns	6=70ns	7=60ns
8=50ns	9=45ns	10=40ns	11=35ns
12=30ns	13=25ns	14=20ns	15=15ns

If you want to learn more, you can consult the Document 31116 AMD Family 10h BKDG Rev. 3.06, register F3xD4, page 240

2.4 Temperature register switches

-temp

This switch will print the actual temperature reported by the internal diode of the processors.

-mtemp

This switch will start a daemon mode which will continously print the temperature reported by the internal diodes.

Note: AMD says that the temperature reported has not to be taken as an absolute real temperature, but instead it is reported only for internal thermal control.

2.5 Hardware Thermal Control switches

-htc

This switch will report useful information about Hardware Thermal Control features of the processors in the system (enabled/disabled, currently active, active in the past, temperature limits, ...)

-htcenable

This switch will enable the HTC feature on active node, so when the processor reach the limit temperature, HTC feature will put it in slow mode to prevent excessive overheating.

-htcdisable

This switch will disable the HTC feature on active node.

-htctemplimit <value>

Sets the high temperature limit on active node as a threshold for thermal control activity. When the processor goes above this limit, the HTC feature will slow down the processor. <value> is indicated in degrees and is usually in the range between 60 and 100.

-htchystlimit <value>

Sets the hysteresis temperature limit on active node. Accepted values here lay between 0 and 7, and they are the degrees the actual temperature has to drop to let the HTC feature being disengaged.

-altvid <vid>

Sets the VID value to use when in Alternate VID mode on active node. Alternate VID mode is used when the processor is overheating, for example.

2.6 PSI bit switches

-psienable

Enable the PSI_L pin control to allow a more efficient power management when the processor is in low power mode, letting the voltage regulators put themselves in a power efficient mode. This applies on active node.

-psidisable

Disable the PSI_L pin on the active node.

-psithreshold <vid>

Sets the threshold VID that asserts the PSI_L pin when the node goes below the specified voltage. It must be a high valued VID because the voltage of the processor has to be low.

Citing the original AMD documentation: *"When the VID code generated by the processor is less than PsiVid (i.e., the VID code is specifying a higher voltage level than the PsiVid-specified voltage level), then PSI_L is high; when the VID code is greater than or equal to PsiVid, PSI_L is driven low "*

2.7 Hypertransport switches

-htstatus

Shows the hypertransport link status, including speed, coherency and width informations for all nodes in the system

-htset <link> <value>

Sets the multiplier of the specified hypertransport link for active node. May cause instability and machine freezing.

2.8 Various and other switches

-c1eenable

Enables the C1E multiprocessing bit for active node/core. To be of any effect, all the cores of a single node must have the bit enabled. **Note:** actually this bit is only officially available for Family 11h processors. It is not described in Family 10h datasheet, but it is reported that some processors of this family have this bit set when C1E is enabled in BIOS and unset when C1E is not enabled in BIOS.

-c1edisable

Disables the C1E multiprocssing bit for active node/core.

2.9 Configuration file switches

-cfgfile <file.cfg>

reads the configuration directly from a text file. For a complete explanation of the various configuration options, just check the example configuration file provided with the program. Also -cfgfile is useful if you want to use the -scaler option, since it allows to personalize the scaler options.

2.10 Performance counter switches

-pcgetinfo

reads some informations about performance counter and show them. Not really useful to the

end user anyway. If you care, take a look to AMD documentation about Performance Counters

-pcgetvalue <counterIndex>
reads the raw value of a performance counter for specified node/core. <counterIndex> parameter has to be between 0 and 3.

2.11 Daemon mode switches

Attention: daemon mode switches must be always the last ones in a command line. Every other switch that follows a daemon mode switch will just be ignored.
To terminate the program while it is in daemon mode, just press CTRL+C.

-scaler
Enables scaler mode. This mode will put the program in daemon mode and the program will take care of processor power management scaling the frequency and voltage of the processor as needed. If you want to personalize the scaler options, you have to use a configuration file. This switch is not compatible with -autorecall switch: the first daemon mode switch will override the other. If you plan to use the scaler mode, you must disable your operating system power management.

-autorecall
Enabled autorecalling mode. This mode will put the program in daemon mode and recalls itself every 60 seconds to refresh the processor configuration. This is particularly useful in Windows when resuming from standby/hibernate state. Actually Linux runs some scripts (everything is put inside /etc/pm/sleep.d folder) when it get back from suspend, so theorically it is not as useful in Linux as it is in Windows. This switch is not compatible with -scaler switch: the first daemon mode switch will override the other. Also take in account that this switch must be the last one, else other switches won't be executed.

-CM
Put the program in Costant Monitoring mode, checking continuosly frequency, voltage and current pstate. This is very useful if you want to monitor your processor transitions and if you want to discover anomalous transitions to pstates over psmax (for example pstates 6/7).

-cpuusage
costantly monitors cpu usage of the processor using performance counters and time stamp counters. This calculates a very precise cpu usage based on processor cycles instead of operating system counters.

Note: command parameters must be issued WITHOUT angle brackets. Angle brackets just mean that a parameter is required for that command switch.

3. Usage examples

Ok, we got the configuration options. Here I propose some little examples to learn the program usage. Here I assume that you haven't overclocked your processor using HT bus tweaking. Actually all here is programmed to assume that you use the default bus frequency. The program will work

fine with overclocked processors, but you may have to do some manual calculations to get coherent frequency values.

3.1 Getting the list of switches

To get the list of program available switches, just run the program from a **command line** with no switches at all, just like this:

```
> TurionPowerControl
```

3.2 Listing the processor features

Usually processors have lots of parameters that are useful for tweaking.

To list some of these interesting parameters there are different options. For example, to list the power states table and take a look to some interesting parameters you can just launch the program this way:

```
> TurionPowerControl –l
```

This will drop out a list of setting about your processor, including the power states table, maximum and minimum VIDs, maximum pstate available, and so on... The table that comes from this command is also very useful if you decide to tweak voltage and frequency because it can be taken as a starting point.

To list some other processor informations, specific for the different families, you can use this switch:

```
> TurionPowerControl –spec
```

Other interesting options are listing current Hardware Thermal Control (HTC) parameters. HTC parameters are enumerated as follows:

```
> TurionPowerControl –htc
```

And you may wish to read the temperature reported from the internal diodes:

```
> TurionPowerControl –temp
```

3.3 Change frequency and voltage (the easy way)

If you want to change voltage or frequency in a easy manner, you have to use the **-set** command switch.

It is really easy to use and pretty user friendly since it will do all the hard job of finding FID, DID and VID values for you. You just have to instruct the program with the frequency or the voltages you want and it will care about the rest.

Let's see a simple example. If you want to set the frequency of your pstate 0 to 2000 Mhz to all cores you just have to launch such a command:

```
> TurionPowerControl -set core all pstate 0 frequency 2000
```

That's it. After this command the pstate 0 of all the cores will receive the right FID and DID values to set their frequency to 2000 Mhz. The program will warn you if there isn't a perfect combination that matches the frequency you requested, and will round the result. Note that you may safely omit the `core all` command because by default all cores are already selected.

Maybe you want to set core voltage too, let's say 1.200 volts, then the command will become like this:

```
> TurionPowerControl -set core all pstate 0 frequency 2000
vcore 1.200
```

Again the program will do all the hard job for you and will warn you if you had choose a voltage that there is no perfect match. Again it will round automatically.

You can also specify different pstates and different settings:

```
> TurionPowerControl -set core all pstate 0 frequency 2000
vcore 1.200 pstate 1 frequency 1000 vcore 1.100 pstate 2 frequency
500 vcore 1.000
```

Such a command will set pstate 0 (to all cores) to frequency 2000 Mhz and voltage 1.200v, pstate 1 to frequency 1000 Mhz and voltage 1.100v and pstate 2 to frequency 500 and voltage 1.000v.

If you want to set voltage core to 1.125 to pstate 0 to just a core (let's say core 1), you can issue this command:

```
> TurionPowerControl -set core 1 pstate 0 vcore 1.125
```

That's all! It's pretty easy, isn't it?

There's the possibility to use **synomims** also, to make the command line more compact. There are the following synonims:

- **pstate** and **ps** have the same meaning
- **frequency**, **freq** and **f** have the same meaning
- **voltage**, **vcore** and **vc** have the same meaning
- **nbvoltage**, **nbvolt**, **nbv** have the same meaning

so we can write more compactly the large example above this way:

```
    >TurionPowerControl –set core all ps 0 f 2000 vc 1.2 ps 1 f
1000 vc 1.100 ps 2 f 500 vc 1.000
```

that is pretty shorter (but also less easy to understand).

Important Note: if your family 10h processor is working in Parallel VID Mode (or PVI for short) instead of Serial VID Mode (SVI), most probably northbridge voltage also controls core voltages too! It means that if you want to change voltage to your cores when running in a specific pstate, you have to change the northbridge voltage. **Explanation:** This is true because PVI mode is enabled when a Dual Plane capable processor (any AM2+ or AM3 processor) is installed in a Single Plane (AM2) motherboard. Single Plane AM2 motherboards can drive only one unified current to the processor for both the cores and the northbridge, instead AM2+/AM3 motherboards can drive two different currents to the processor, one for the northbridge and one for the cores.

3.4 Change the northbridge voltage (the lazy way)

The **lazy and easy way** is to use the usual -set switch.

It differs slightly between family 10h processors and family 11h processors. Family 10h processors have northbridge voltage per-pstate, so each pstate has its own northbridge voltage. An example to control it follows here:

```
    >TurionPowerControl –set pstate 3 nbvoltage 1.10 pstate 4
nbvoltage 1.05
```

The command above will instruct the program to set to pstate 3 the northbridge voltage of 1.100 volts and to pstate 4 the northbridge voltage of 1.050 volts. This value is forced to **all** cores automatically as it is stated in AMD official documentation.

Important Note: if your family 10h processor is working in Parallel VID Mode (or PVI for short), most probably northbridge voltage also controls core voltages too! It means that if you want to change voltage to your cores when running in a specific pstate, you have to change the northbridge

voltage. In such a situation, changing pstate core voltages simpy has no effect at all.

Explanation: This is true because PVI mode is enabled when a Dual Plane capable processor (any AM2+ or AM3 processor) is installed in a Single Plane (AM2) motherboard. Single Plane AM2 motherboards can drive only one unified current to the processor for both the cores and the northbridge, instead AM2+/AM3 motherboards can drive two different currents to the processor, one for the northbridge and one for the cores.

On Family 11h processors there is a unique northbridge voltage for all pstates, so you can put **nbvoltage** keyword wherever you want, even without specifying any pstate. And example of this is following:

```
>TurionPowerControl –set nbvoltage 0.900
```

Another example that will do exactly the same is

```
>TurionPowerControl –set pstate 2 nbvoltage 0.900
```

3.5 Tweaking multiprocessor machines

If you have a multiprocessor machine (i.e. a machine with more than one physical processor, not a machine with processor with multiple cores), you may like to tweak all the processors (or **nodes**) in a single shot. As usual, you can use the **-set** switch to tweak your parameters. You can use the **nodes** command to specify that the tweak you're going to apply will affect all nodes or just a precise node in the system.

Going back to the example of paragraph 3.3, we want to set for all cores of node 2 in the system the frequency of 2000 Mhz to pstate 0. This command will be useful:

```
> TurionPowerControl –set node 2 core all pstate 0 frequency
2000
```

Node command should be followed by an integer if you wish to specify the node, or by *all* word to specify that the tweak will affect all the nodes. If you omit the node command, the program will just assume that you're going to tweak all the nodes (just as like as if you omit the core command the program assumes that you're going to tweak all the cores).

If you wish to set the voltage of pstate 0 of all nodes of all cores you can issue a command like this:

```
> TurionPowerControl –set node all core all pstate 0 voltage
1.100
```

Such a command will set 1.100 volts to all the cores of all nodes in your multiprocessor machine. Since by default all nodes and all cores are selected, you may write the following command that has the same behaviour of previous command:

```
> TurionPowerControl –set pstate 0 voltage 1.100
```

3.6 Force a transition to specific pstate

Well, normally you won't care about telling the processor(s) to go into a specific pstate because the operating system will care itself, but if you need to do such a job you can use the **-fo** switch.

For example, to force all cores of all processors in the system to switch to power state 1, you can issue a command like this:

```
> TurionPowerControl –node all –core all –fo 1
```

Since all nodes and all cores are activated by default, you can accomplish the same task of above executing this command:

```
> TurionPowerControl –fo 1
```

Of course you can use a finer granularity. If you want to force power state 0 for core 2 of node 1 and power state 2 for core 3 of node 1 just write:

```
> TurionPowerControl –node 1 –core 2 –fo 0 –core 3 –fo 2
```

Just remember that once you activate (with **-node** or **-core** switch) a node or a core, it will remain active until you explicitly activate something else.

3.7 Using the cpu usage meter

TurionPowerControl implements the most precise cpu usage counter available for AMD processors. It is the most precise simply because it uses processor specific performance counters.

To use the performance usage meter, just issue:

```
> TurionPowerControl –perf-cpuusage
```

and it will go into daemon mode telling you every second the actual cpu usage of every core. To exit the program while in daemon mode you have to press CTRL-C.

Note: if you have a processor with Turbo Core functionality, TurionPowerControl may tell you that some cores are loaded for more than 100%. This is normal and expected. It actually shows you that

the Turbo Core functionality is really working on those cores. Also this applies if you overclock your processor.

3.8 Using the autorecall feature (useful to restore parameters after hibernation/suspend)

The autorecall feature is useful to automatically restore the parameters after the computer goes into suspend or hibernation.

Using the autorecall feature is very simple, just issue your preferred command following the **-autorecall** switch at last.

For example:

```
> TurionPowerControl –set core all pstate 0 vcore 1.125 –autorecall
```

This command will execute the -set command and then will put itself in autorecall mode. It means that every minute will automatically re-execute the commands on the command line. Autorecall is a daemon mode, so to stop the program you have to push CTRL-C.

3.9 Startup hints for Windows and Linux

If you use Windows it would be useful to run the program at startup and use the autorecall switch to let it restore automatically the settings after your computer has been in suspend or hibernation.

You can follow many guides on the internet that uses the Windows Vista/7 **Task Scheduler** to allow programs to be run at startup with **administrative privileges** (it is important). One of such guides is available here (K10STAT AMD Griffin Processor UnderVolting Guide by weinter).

On Linux you can put a startup command in the common **/etc/rc.local** file. This file will be automatically loaded at startup by modern distributions (don't use autorecall there, or you system won't load until you kill the daemon!). Resuming from suspend and hibernate is not a problem on Linux, since the kernel is instructed to run every script is laying in **/etc/pm/sleep.d/** folder everytime the computer wake up.

3.10 How to obtain some more pstates

Ok people, this is a very experimental trick and I can't guarantee it works flawlessy. If you feel in the right mood, you may want to give it a try.

Processors of Family 11h (mobile Turions and Athlons) come with **8 pstates**, but normally just **2 or 3** of them **are enabled**.

Processors of Family 10h (desktop Phenoms and Athlon Iis) come with **5 pstates** and they got a

variable number of enabled pstates. Older 65nm Phenoms (first series, the hot ones) have only **2 pstates enabled**, instead newer 45nm Phenom II and Athlon II have 4 or 5 pstates enabled.

This trick is useful for people who has a limited number of enabled pstates. If you already have 4 or 5 pstates enabled, I don't think it would be of any interest.

Anyway, to try to make usable some disabled pstates you have to follow this guide:

- First of all, you need to disable your operating system CPU frequency scaler. You may disable it just telling your operating system that the processor has to stay at a fixed frequency.

- Then you have to **enable** the pstates you want to "unlock". Let's make an example, you got a single processor machine with a four core processor with two pstates already enabled (pstates 0 and 1) and you want to unlock a third (pstate 2). You may run such a command:

  ```
  >TurionPowerControl –core all –en 2
  ```

  This way you are going to enable pstate 2 on all the cores of the processor.

- Now just put some valid frequency and voltage values inside the pstates you just enabled. An example of this:

  ```
  >TurionPowerControl –set pstate 2 core all frequency 800
  voltage 0.900 nbvoltage 1.150
  ```

  of course this is an example. As stated in the official documentation, the higher the pstate, the lower the frequency and voltage have to be. Also, if you're tricking on a Family 11h processor you should remove the nbvoltage statement.

- Then instruct the processor to change its maximum pstate. If you had 2 pstates enabled, probably your maximum pstate value was 1. Now that you have 3 pstates enabled, you have to set your maximum pstate to 2. Such a command will do the job:

  ```
  >TurionPowerControl –psmax 2
  ```

- Now you have to run the TurionPowerControl scaler. This is a daemon mode described in **Paragaph 2.10** and will take care of frequency and voltage switching. There are some parameters that can be personalized, but you need to use a configuration file (see **paragraph 2.8**). Anyway if you wish to run the scaler with default parameters just issue:

  ```
  >TurionPowerControl –scaler
  ```

- Maybe you may want to check if the processor is scaling correctly, so you can run another command line sheel and then run the Constant Monitor mode:

  ```
  >TurionPowerControl –CM
  ```

  to see if it effectively is switching to the pstates you unlocked.

4. Advanced Usage and formulas

Note: you can freely skip this whole chapter if you are not really interested in the knowledge about how the calculations for pstates are done and how to tweak the processor pstates with finest granularity. Stick to Chapter 3 usage examples if you are a standard user that don't mind those awful technical things described here.

4.1 Family 10h processors formulas (Phenom, Phenom II, Athlon II and mobile Turion II processors)

This is the formula to obtain the actual frequency from FID and DID parameters:

```
freq = (100*(Fid+16))/(2^Did)
```

For example, in pstate 0 I have FID = 14 and DID = 0 so

```
freq = (100*(30))/(2^0) = 3000 Mhz
```

VID determines the actual core voltage for each pstate and is calculated in two ways. Despite the fact that there are two voltage modes (Parallel VID Mode and Serial VID Mode), the processor will just store their VID using a 7 bit register. That means that VID can have values ranging from 0 to 127. To translate VID to vcore value you can use this function:

```
vcore = 1,55 – (VID * 0.0125)
```

On the hardware side note that if your motherboard is using Serial VID Mode, the voltage requested by the processor is exactly that one coming from the formula above. Instead if your motherboard is using Parallel VID Mode (AM2+/AM3 processor on AM2 motherboard) the real vcore requested by the processor is approximated.

4.2 Family 11h processors formulas (Turion ZM/RM and Athlon QL processors)

This is the formula to obtain the actual frequency from FID and DID parameters:

```
freq = (100*(Fid+8))/(2^Did)
```

For example, in pstate 0 I have FID = 13 and DID = 0 so

```
freq = (100*(21))/(2^0) = 2100 Mhz
```

VID determines the actual core voltage for each pstate and is calculated in this way:

```
vcore = 1,55 – (VID * 0.0125)
```

Again, my VID at pstate 0 is 36, so vcore is 1.100 volt.

Voltage regulator register is 7 bit wide, so normally it could represent and integer between 0 and 127, but actually it is limited internally by some hardwired settings: unfortunately VID cannot exceed the value 64 (vcore 0.750v) for Turion RM and ZM processors and 52 (vcore 0.900v) for Athlon QL processors, so there is a lower voltage limit we can't pass over. There is also an upper limit set in the software, that is 28 (= 1.200 volts).

Differences between ZM, RM and QL processors:

ZM processors are most adavanced, they have three power planes (one for northbridge and two per each core), so core voltage can be varied independently per core.

RM processors are pretty similar to ZM processor, but they have 512kb L2 cache per core and have just two power planes, one for northbridge and one for cores. This means that core voltage is supplied to both cores and is determined by the highest core voltage of the core in the highest pstate.

QL processors are a minor class (they actually are Athlons, not Turions) and have two power planes. The particularly with these processors is that they have just two pstates, and not three like Turions. This reduces power savings, but actually they share the same constructive goodness of Turion models, so their core voltage can be reduced to reduce heating and prolong battery life. Also latest QL models (expecially QL-64/65/66 models) looks promising about undervolting at fastest pstate. Some users reported that they can easily go below 1.000v for pstate 0 at full frequency!

4.3 Change voltage and frequency (the hard way)

To change the voltage or the frequency of a pstate for all the cores, you can use still use the **-set** swtich but using **fid, did** and **vid** commands. This will allow you a finer control and is also more powerful because it can allow you to set exactly the parameters you want, but you need to make some calculations previously. If you don't want to mess with some (simple anyway) math, I suggest to watch the examples of the **paragraph 3.3**.

Issuing a command like:

```
> TurionPowerControl -set pstate 1 vid 48 did 1 fid 10
```

Will set (for all cores at once) pstate 1 to have VID=48, DID=1 and FID=10.

What this command actually does depends on your processor and it's VID mode configuration:

- If you get a **Family 10h** voltage will be **0.950v** and frequency will be **2600 Mhz**
- If you get a **Family 11h** processor, voltage will be **0.950v** and frequency will be **900 Mhz**

To obtain all these numbers, you can refer to Chapter 4 about the formulas.

Remember also that you have a Family 10h processor if you have a Phenom, a Phenom II, an Athlon II or a Turion II processor, and you got a Family 11h processor if you have a mobile Turion ZM/RM or Athlon QL processor.

Look also that you don't need to run several times the program to change multiple options. You can just run the program once and concatenate the switches this way:

```
> TurionPowerControl –set pstate 1 vid 48 did 1 fid 10 pstate 2 vid 60 did 2 fid 8
```

4.4 Change the northbridge voltage (the hard way)

The hard way is to use the -nbvid switch. Again there's a distinction between family 10h and family 11h processors (see **paragraph 3.4** above). On family 10h processors it will accept two parameters: the pstate you want to modify and the northbridge VID. Northbridge VID can be calculated using the usual formulas in chapter 4. This is an example:

```
> TurionPowerControl –nbvid 2 36 –nbvid 3 40
```

This way you set Northbridge VID to 36 for pstate 2 and to 40 to pstate 3. As you need to use the formulas coming from chapter 4 you need also to take care about Parallel VID mode and Serial VID mode. Since this is a bit annoying, I suggest you to use the **-set** switch that will take care itself of this (see **paragraph 3.4** above, again).

On family 11h processors, northbridge voltage is unique for all pstates and all cores. If you own a family 11h processor, the -nbvid switch will accept just one parameter which is the VID you want to set. Again formulas to translate from VID and voltage and viceversa are in Chapter 4.

An example of this follows:

```
> TurionPowerControl –nbvid 48
```

This will set Northbridge VID to 48, that is 0.950v according to formulas. As above, if you want an easier way to set northbridge voltage, you can always use the -set command described in paragraph 3.4

4.5 Change frequency with maximum granularity (the hardest way!)

The program allows you to change the voltage and frequency with finest granularity. You can change, for example, the frequency of a single pstate of a single core. To do this you need to use **-set** switch and use the simple formulas you can obtain in Chapter 4.

For example, you got a **Family 11h** processor and want to change the operating frequency in pstate 1 of core 0 to 1000 Mhz. From the formula you get that FID must be 12 and DID must be 1, so issuing:

```
> TurionPowerControl –set core 0 pstate 0 did 1 fid 12
```

will set FID of core 0 pstate 0 to 12 and will set DID of core 0 pstate 0 to 1, obtaining a final frequency of 1000 Mhz

If you got a **Family 10h** processor, setting the same frequency of 1000 Mhz to Core 0 and pstate 1 is obtained setting FID to value 2 and DID to value 1, so here it the command:

```
> TurionPowerControl –set core 0 pstate 1 fid 2 did 1
```

4.6 Change voltage with maximum granularity (are you crazy enough?!?)

This tweak has a sense only if you got a Turion ZM processor (Family 11h), since they have a triple plane design, where every core has its own power plane and then the northbridge has another plane too. These processors can vary in a total independent way frequency and voltage, were Family 10h processors can't (with some minor precisations).

Anyway to change the voltage of a specific core and of a specific pstate you can use the **-set** switch, like this way:

```
> TurionPowerControl –set core 1 pstate 2 vid 60
```

This command will set the VID of core 1 of the pstate 2 to the value of 60. The actual voltage that will be feed to the processor will be **0.800v**.

5. Compiling

5.1 How to compile with Linux

I ported the code using gcc and Fedora Linux Core 10 x64, with kernel 2.6.27.21-170.2.56.fc10.x86_64.

Actually it has been compiled by myself and ran well on Fedora Linux Core 11 and 13 x64 with

2.6.31, 2.6.32, 2.6.33 and 2.6.34 kernels. Other people compiled well on many distros, including Ubuntu, Slackware, OpenSuse and ArchLinux.

There should be no problems compiling on i386 targets since if doesn't use any specific code, but check **Paragraph 1.3 - Linux Requirements** for more details.

To compile the program for your specific Linux distribution, just use **make** command to allow automatic compilation.

To compile just put yourself in the source code folder and issue this command:

```
> make
```

At the end of compilation, if everything went good, you'll find an executable file called `TurionPowerControl` in the same folder of source code.

If you wish to install the executable inside the **/usr/bin** folder, issue these commands (for Fedora, Suse, and other non-Ubuntu based distributions):

```
> su
```

then write your root password, then write:

```
> make install
```

If you are using Debian/Ubuntu based distributions:

```
> sudo make install
```

Then you will be able to reach the program for everywhere in your filesystem.

If you find troubles or strange errors when you launch the software, keep in mind you have to be **superuser** to access and manipulate processor features.

You may also encounter errors about retrieving **CPUID** informations or access to **cpu MSRs**, so be sure that your system has the required **cpuid** and **cpumsr** modules loaded (if they are not yet compiled inside your kernel). This is most common on Ubuntu distributions. You can load them issuing (always as a superuser):

```
sudo modprobe cpuid
sudo modprobe msr
```

If you still can't load these modules, check that your distribution come with them, else I think you can download them using your distribution favourite package manager. Again, check **Paragraph 1.3 – Linux requirements** if you have troubles.

6. Contacts and donations

For any question feel free to send me an email to: blackshard@email.it