ISSN 2277-3061

The Effects of Various Frequency Scaling Algorithm on Embedded Linux CPU Power Consumption

D. Shuhaizar[1], R. Badlishah Ahmad[2], Ong Bi Lynn[3]
[1,2,3] Embedded Computing Research Cluster,
Unit Kluster Penyelidikan Universiti Malaysia Perlis,
Jalan Pangkalan Assam, 01000 Kangar, Perlis, MALAYSIA
[1]shuhaizar@gmail.com
[2]badli@unimap.edu.my
[3]drlynn@unimap.edu.my

ABSTRACT

In this paper we have tested various frequency scaling algorithm available in Linux and measured the effects of the different algorithms during operation in an actual system. We have tested all the default scaling algorithms included in Linux in a controlled testing environment and measured the actual effects of the scaling algorithms to the CPU power consumption during operation. In order to retain high accuracy and avoid unnecessary components from affecting the results, we have conducted the measurement directly on the CPU power supply line and logged the power usage in real time.

Indexing terms/Keywords

Embedded Systems; Dynamic Voltage & Frequency Scaling; Power Consumption; Embedded Linux

Academic Discipline And Sub-Disciplines

Computer Technology; Embedded Systems

SUBJECT CLASSIFICATION

Embedded system, energy saving.

TYPE (METHOD/APPROACH)

Power measurement; Experimental setup; On circuit power measurement

ISSN 2277-3061

INTRODUCTION

Linux is fast becoming a major player in the embedded devices platform. With projections from research firms suggesting that shipping of Internet connected devices and Internet of Things (IOT) embedded devices surpassing the shipment of personal computers and smartphones, it is more imperative that maximum energy efficiency are extracted from the devices so that operational time of the devices are prolonged [1].

With the advent of multicore processors and cheaper costs to integrate them into an embedded device, more and more manufacturers are using such processor and solution in their final product. In this paper we take a look at the various frequency scaling algorithms available in an embedded linux platform and benchmarked them to see their effects to the system power consumption in a controlled test condition.

DYNAMIC VOLTAGE & FREQUENCY SCALING

The concept of reducing the voltage and frequency during operation as an energy saving technique was first outlined in a research paper published by Weiser et.all in 1994 [2],[3]. Dynamic Voltage & Frequency Scaling (DVFS) is a method introduced in line with the research to reduce a processor's operational frequency on the fly according to load. The motivation behind DVFS is that reduction in the processor frequency would allow a reduced power usage by the processor. This in theory would bring down the system power consumption during idling and in operation. This also helps to prolong the lifespan of the CPU by reducing heat dissipation thus avoiding premature failure [4],[5],[6]. Such a technology is particularly beneficial for embedded systems mainly because of the power constraints and heat dissipation reduction that allow for better use of the available power [4]. The concept of reduced power consumption in an embedded system especially battery power devices are categorized as more important than their fixed power system with access to wall power [1].

In Linux the DVFS function is provided by the *cpufreq* interface for processors supporting such function in their hardware mechanism [7],[8].

LINUX CPU THROTTLING MECHANISM

Linux in itself supports various CPU scaling algorithm ranging from a fixed execution frequency to load based scaling algorithm. The kernel component responsible for the scaling process is implemented in the cpufreq interface placed in the /sys/devices/system/cpu directory [8]. In default configuration, linux supports 5 different CPU scaling algorithm called governors [9]. The governors available for use are:

- Userspace - user set, userspace specified frequency, locked according to user request. Used to provide the low frequency and high frequency lock in this test.
- Performance - CPU statically set to the highest frequency within the borders of scaling_min_freq and scaling_max_freq.
- Powersave - CPU statically set to the lowest frequency within the borders of scaling_min_freq and scaling_max_freq.
- Ondemand - Set the CPU frequency according to the CPU load.
- Conservative - Set the CPU frequency accordint to CPU load, gradual increase & decrease.

Although introduced in late 2004, the ondemand governor is still considered as the best CPU scaling algorithm available in Linux [7],[10]. Development of the ondemand governor are based on the assumption that the power requirement of CPU scales according to the current clock frequency of the processor [11],[12],[1]. The development of the governor itself are made around processor technology available back in 2004 and since then numerous technological advancement have been made in CMOS and processor technology and processor power management solutions. In this paper we tested the various cpu governors and measure their power performance in real time during operation.

CPU POWER PREDICTION

Past research have found that the power consumed by a CPU during operation is linear to the CPU clock frequency [11],[12],[1]. Simulation based power measurement prediction assumes that the CPU power consumption is calculated as the cube of frequency, which do not take into account the processor's idle power requirement [11]. Simulated power consumption assumes that power consumption of the CPU is calculated with the equation below:

$$P_{active} = S_3 \bullet f^3$$,where P is the active power consumption, S3 is a constant and f is the frequency (Equation 1).

This equation depends heavily on the CPU frequency to be a major contributor to the active power consumption. According to the equation, reduction in the clock frequency will greatly affect the active power. Another generally used equation to predict CPU power consumption comes in the form of CMOS power usage equation given below:

$$P = CV^2 \bullet f$$,where P is power consumed, C is capacitance, V is supply voltage and f is the frequency (Equation 2).

Equation 2 depends heavily on CPU core voltage to affect the total power consumed. Since the CPU supply voltage is squared, any changes to supply voltage will have the biggest effect to the power consumption. Both equations do not consider the idle power requirement which is usually much smaller compared to the active state [11]. During idle state, modern CPU have significantly smaller power requirement compared to during heavy operation. Failure to take this into consideration leads to a very different between predicted and actual measured power consumption. While both equations

provide sufficient precision in predicting the active state power consumption, this is usually not the case during for actual operation whereby the processor would be sitting idly while waiting for incoming jobs [11]. This is particularly true for an embedded system where the processor would be waiting in idle mode most of the time. In this paper we measured the actual power requirement of modern embedded processors working with an actual load and log the power consumption in real time.

POWER MEASUREMENT SETUP

Hardware Setup

In order to get the highest accuracy of power measurement, the actual power consumption of the test platform must be carried out during operation in real time. Instantaneous current draw and CPU vcore supply must be measured and logged during the testing procedure. To avoid unnecessary component such as the network controller and display adapter from interfering with the measurement and affecting CPU power measurement accuracy [7], power consumption data have to be logged at the CPU power supply line. CPU power measurement setup is given in the figure below:

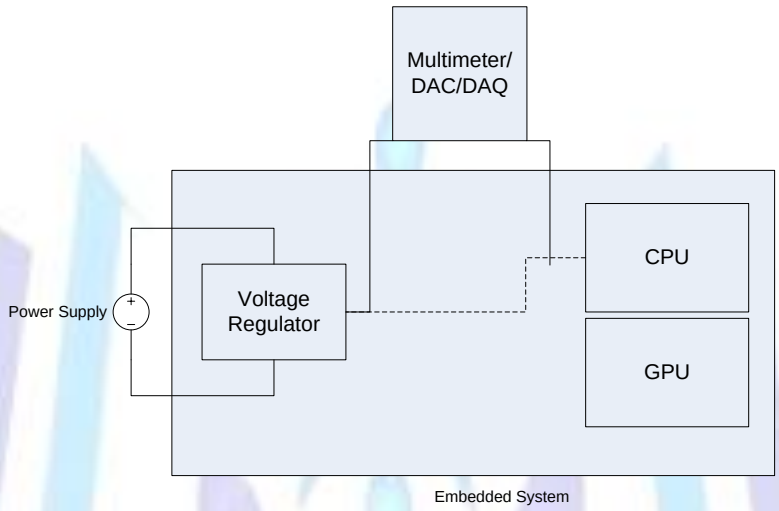

Fig 1: CPU power measurement setup.

A modern CPU developed particularly for embedded systems consisting of the Intel Atom N2800 multicore CPU have been chosen for implementation. The embedded platform consisting of an Intel Desktop Board DN2800MT is used. The board follows mini-ITX specification and powered by a single DC power supply without any active cooling [13],[14]. The CPU supports 5 different frequency states (0.798, 1.064, 1.33, 1.596 and 1.862 Ghz) which could be manipulated through the *cpufreq* interface [13]. The processor also has 2 different cores which could be manipulated differently from each other.

Software Setup

To provide a load algorithm, a library compilation procedure has been chosen as the workload as it provides a random load mimicking actual real life situation. The load selected is the *torch* library compilation procedure. We have measured the library compilation sequence from start to finish and the power load during the procedure are observed and logged.

ISSN 2277-3061

CPU POWER MEASUREMENT

Entire Compilation Sequence

We have measured the entire torch library compilation sequence and the results are as follow:

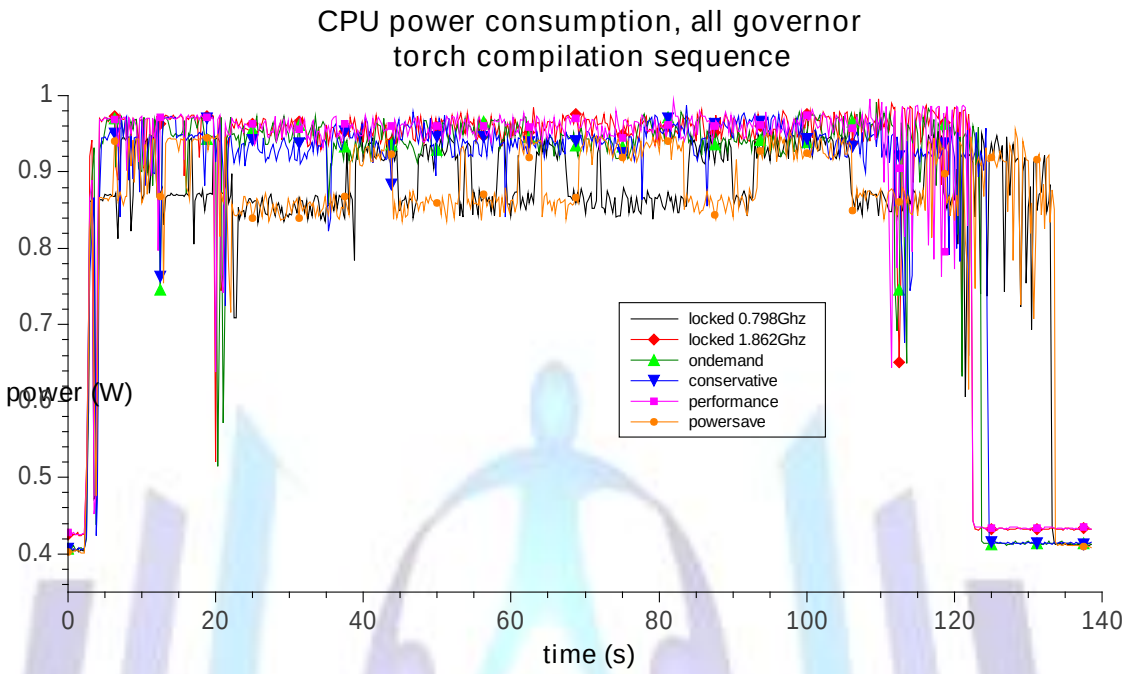

Fig 2: CPU power consumption, entire sequence

From the entire graph we can see a clear pattern during the logging sequence. While all the governor starts at the same time, the *powersave* and locked low speed required more time to finish the compilation process. The 2 slower governor finish the compilation process nearly 10 second slower compared to the rest of the system. Whilst being slower, the 2 governors however consistently requires less power during the compilation stage (0.86W versus 0.96W). This is around 10% lower power compared to the faster governors.

Run-to-idle Performance, End Compilation Sequence

Zooming at the end of the compilation sequence we can see this pattern:

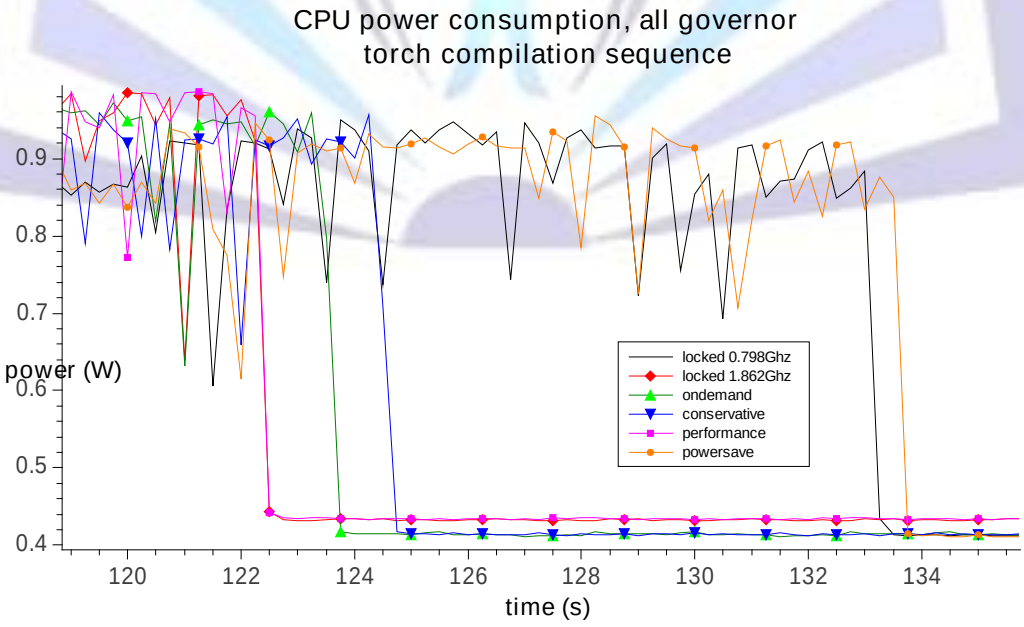

Fig 3: CPU power consumption, run-to-idle at end of test sequence.

The 2 slower governors (*powersave* and low speed lock) are apparently around 10 seconds slower compared to the faster governors. We can see that the *performance* governor finishes the compilation sequence at the same time as the locked high frequency governor. *Ondemand* finishes 1.25 second and *conservative* finish 2.25 second slower compared to the performance oriented governors. This shows that performance oriented scaling algorithm finishes faster compared to energy-oriented algorithms thus providing the system with a faster run-to-idle performance. However, even executing the sequence at more than double the clock frequency (0.798GHz versus 1.862GHz), the performance gain are not linear and we only improved the execution performance by around 8%. We cannot expect that doubling the clock frequency will translate to double the execution speed as shown in the result. This is similar to the finding proposed by [15] mentioned in [3].

Start & End Idling Power

While the performance oriented CPU governors (performance & locked high frequency) finished the compilation process faster and have a faster run-to-idle speed, the idling power utilizing these governors are far from ideal. Zooming at the start of the compilation process where the processor are idle, we have this graph:

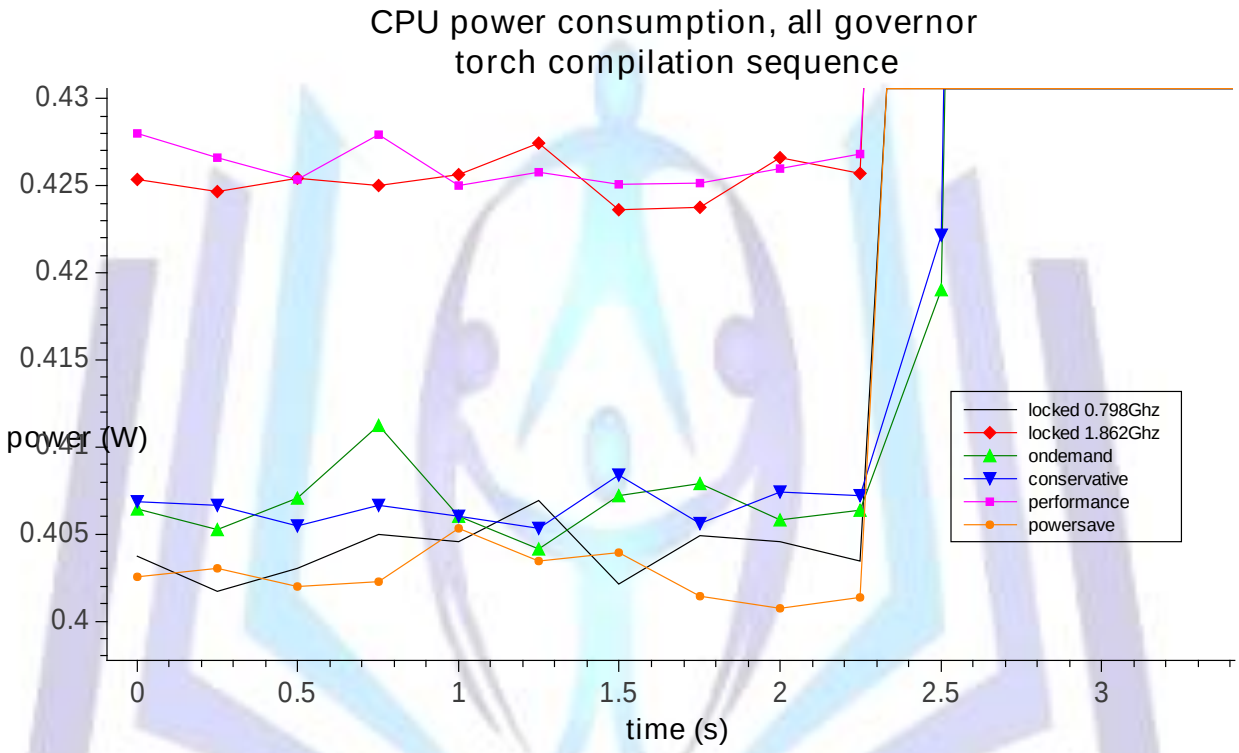

Fig 4: CPU idling at start of test sequence.

At the start of the compilation sequence, the performance-oriented governors (*performance* and locked highest speed) idles at a higher power compared to the rest of the group. The performance and high speed governor idles at 0.425W while the rest of the group have an average of 0.405W in idle. The difference translates to around 4.9% higher power used by the performance-oriented governors. The lowest power observed is from the *powersave* and low speed locked which on average idles at 0.403W.

We observed similar pattern for the idling at the end of the test procedure as graphed in the figure below:

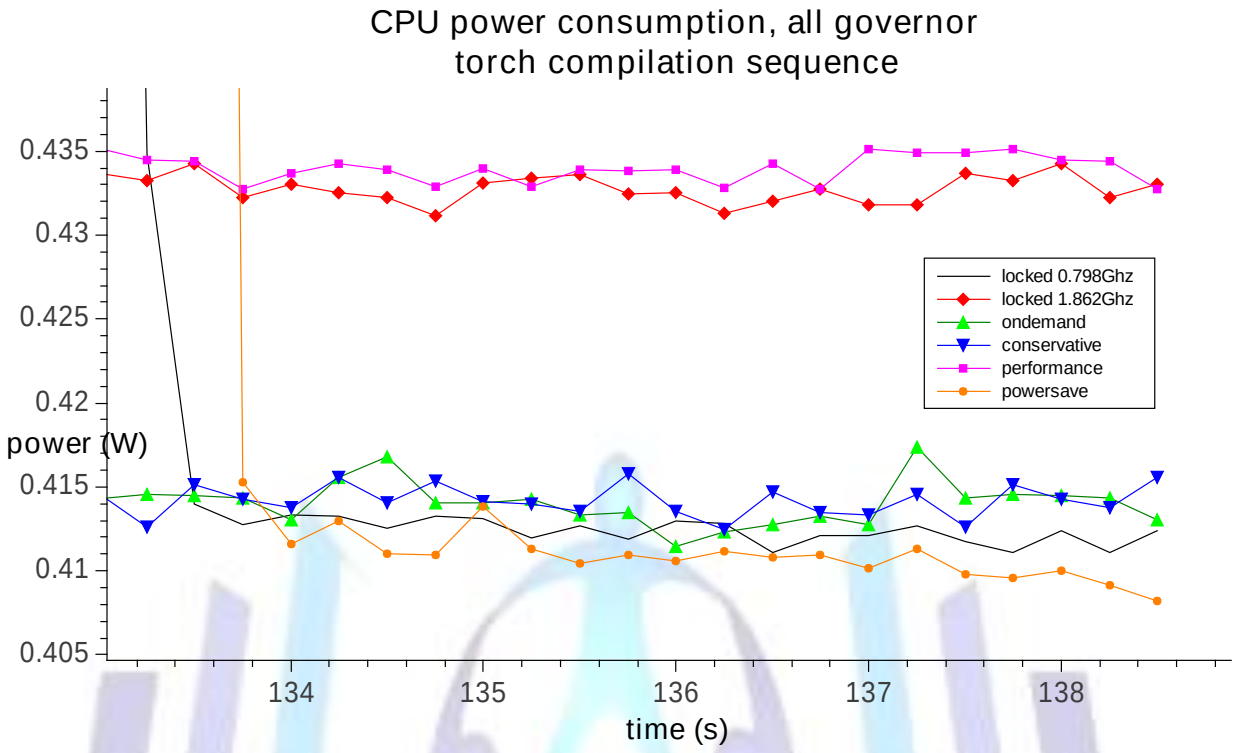

Fig 5: CPU idling power at end of test sequence

At the end of the test procedure we observed similar pattern as above where the lowest power consumption is from the *powersave* governor. The *ondemand* governor, *conservative* and low frequency lock idles a bit higher but is still far lower compared to the idling power of *performance* and high speed lock. On average the high power group idles at 0.433W while the low power group idles at 0.413W. The difference is around 4.84% which is not much but taking into account that the idle power will have prolonged effect on the system battery and operational life it makes more sense to have a lower power in idle.

CONCLUSION

From the tests that we have carried out, we can conclude that there is an energy saving to be had by allowing a processor to idle at a lower frequency. While a difference of around 4.8% is not significant, considering that the idling time would be very long during operation, it makes much sense to allow the processor to idle at a lower speed. We also came to the same conclusion as previous studies where the gain from increasing clockspeed are not translated to a linear performance gain in execution whereby in our test an increased to double the clockspeed only translated to roughly 8% improvement in actual execution speed. Our measurement have shown that a dynamic frequency scaling algorithm such as *ondemand* and *conservative* provides the system with balance of better execution speed for faster run-to-idle whilst maintaining a low power consumption and better energy usage during idling.

REFERENCES

[1] H. Kimm, S. Shin, and C. O. Sung, "Evaluation of interval-based dynamic voltage scaling algorithms on mobile Linux system," presented at the Proceedings of the 2007 ACM symposium on Applied computing, Seoul, Korea, 2007.

[2] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," presented at the Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation, Monterey, California, 1994.

[3] D. C. Snowdon, S. M. Petters, and G. Heiser, "Accurate on-line prediction of processor and memoryenergy usage under voltage scaling," presented at the Proceedings of the 7th ACM & IEEE international conference on Embedded software, Salzburg, Austria, 2007.

[4] M. Kadin and S. Reda, "Frequency and voltage planning for multi-core processors under thermal constraints," in *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, 2008, pp. 463-470.

[5] R. Ayoub, U. Ogras, E. Gorbatov, J. Yanqin, T. Kam, P. Diefenbaugh*, et al.*, "OS-level power minimization under tight performance constraints in general purpose systems," in *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, 2011, pp. 321-326.

[6] B. Sander, J. Schnerr, and O. Bringmann, "ESL power analysis of embedded processors for temperature and reliability estimations," presented at the Proceedings of the 7th IEEE/ACM international conference on Hardware/software codesign and system synthesis, Grenoble, France, 2009.

[7] V. Pallipadi and A. Starikovskiy, "The Ondemand Governor: Past, Present, and Future," in *Linux Symposium*, Ottawa, Ontario, Canada, 2006.

[8] I. Corporation, *Enhanced Intel® SpeedStep® Technology for the Intel® Pentium® M Processor*: Intel Corporation, 2004.

[9] D. Brodowski and N. Golde, "CPU frequency and voltage scaling code in the Linux(TM) kernel," in *Kernel.org Documentation Project*, ed.

[10] D. Domingo, "Power Management Guide," in *Performance Tuning Whitepapers*, ed: Red Hat Inc.

[11] S. Saha and B. Ravindran, "An experimental evaluation of real-time DVFS scheduling algorithms," presented at the Proceedings of the 5th Annual International Systems and Storage Conference, Haifa, Israel, 2012.

[12] M. P. Lawitzky, D. C. Snowdon, and S. M. Petters, "Integrating Real Time and Power Management in a Real System," in *Fourth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, Prague, Czech Republic, 2008.

[13] I. Corporation, "Intel® Desktop Board DN2800MT- Technical Product Specification," ed, 2012, p. 107.

[14] I. Corporation, "Intel® Desktop Board DN2800MT- Product Guide," ed: Intel Corporation, 2012, p. 62.

[15] T. L. Martin and D. P. Siewiorek, "Nonideal battery and main memory effects on CPU speed-setting for low power," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 9, pp. 29-34, 2001.