GreenIT: Power Consumption and -Optimization of PCs

Holger Macht

<holger@homac.de>

July 17, 2008
Revision: 0.1.1

Contents

1 Introduction

GreenIT, a buzzword that can be found almost everywhere these days. It is a movement to ensure environmental awareness throughout the computer industry. It contains efforts to keep technologies and electronic devices environment-friendly over their whole live cycle, including design, production, use and disposal. Beside avoiding harmful substances in production and wondering about the ability to have recyclable components, it also includes system designs, which animate operating system vendors to strengthen their efforts in the power management area.

Having environmental protection as a common goal that should be of interest for everyone, end users, customers, the industry as well as governments, everybody tries to reduce the greenhouse effect and not to pollute the environment with harmful chemicals or electronic waste.

But also increasing energy prices and general lack of resources force companies to look for energy saving solutions. They are jumping at the chance to reduce their power costs, as well as getting good marketing possibilities as a side effect.

However, home and business users have another completely different reason for supporting the *GreenIT* approach. They want their laptops, PDAs and all other kind of mobile devices to run as long as possible, without the need of an external power supply. Things like surfing the web while sitting in a coffee shop and "Computing on the go" becomes more and more self-evident.

Being aware of the facts why *GreenIT* makes sense, this document will consider computer systems in use, leaving production and recycling aside. After giving a little bit of background information, it will focus on the ACPI specification as one of the most important parts and the basis of computer power management from an operating system's point of view. After describing a computer system as a unit, some selected hardware components will be considered in regard to their power saving possibilities and how operating systems can make use of them. The last section will cover strategies about how the power saving features can be used in regard to user behaviour and not to disturb normal computer operation. As an initiation into the topic, it now starts with a small excursion into history.

2 ACPI

2.1 Advanced Power Management (APM)

In former times, APM, the so called "Advanced Power Management" developed by Intel and Microsoft has been the de facto standard for power management capabilities typical laptop and desktop computers expose. Within this design approach, the hardware and BIOS itself was in charge of controlling and monitoring the power management functionality, bypassing the operating system.

This included one big advantage. Operating systems running on that hardware did not have to care at all, or only to a very low extend. Manufacturers and BIOS designers

had to pay attention that their implementations work, and if that was the case, the hardware worked no matter which operating system was in place.

However, it contained some drawbacks, too. Different hardware manufacturers are designing and implementing the same functionality over and over again. They did not only have to care that their hardware works from a technical point of view, but also from a policy point of view. Different use cases of different user and target groups have caused this design to become very inflexibly. Imagine a network card deployed in a data server or in a laptop. In server area, performance might take precedence over power consumption, whereas a laptop user wants his battery to last as long as possible, taking a loss in performance. This turned out to be the major problem of APM.

The prevailing disadvantage finally caused this standard to lose more and more importance and acceptance in the industry. Nowadays, APM nearly vanished completely from recent systems.

Obviously, system architectures cannot go without a proper standard for power management related tasks. Having the problems of APM in mind, industry leading companies like Hewlett-Packard, Intel, Microsoft, Phoenix Technologies and Toshiba had to sit together to develop something new. As an outcome, ACPI was born.

2.2 Advanced Configuration and Power Interface (ACPI)

ACPI is an open industry standard first published in 1996. Until now, its specification contains about 611 pages which are kept quite general, making it more extensive than the former APM.

The specification as a whole cares about a lot more than just power management. It can be divided into two major logical parts. One *Configuration Interface*, and one *Power Interface*. The former contains control methods for communication with the hardware and for things like “Plug and Play”. The latter cares about system-, sleep- and device power states. In this document, only the *Power Interface* will be of interest.

ACPI defines common interfaces which are equal across hardware from different manufacturers. This time, the operating system is in charge of controlling the hardware, making the approach a lot more flexible. The manufacturers just have to implement and provide the interfaces defined in the specification giving full control to the operating system. Different driver implementations can take care of different use cases.

If the operating system needs to take control, it needs to know a lot of details about the hardware, about which interfaces are provided and so on. To accomplish that, the vendors ship so called *System Description Tables* along with the hardware and the BIOS.

2.2.1 ACPI System Description Tables

The system description tables describe the interfaces to the hardware. Common examples are the DSDT, FADT, SSDT, etc... The tables have a hierarchical structure and contain so called “definition blocks” for each type of component. They are written in AML (ACPI Machine Language) and are decoded by the operating system with an interpreter. An example of how an excerpt of a DSDT can look like is provided below:

```
Scope (\_SB)
{
    Device (GDCK)
    {
        Name (_HID, EisaId ("IBM0079"))
    [...]
        Method (_DCK, 1, NotSerialized)
        {
    [...]
        Method (_EJ0, 1, NotSerialized)
        {
    [...]
        Method (_STA, 0, NotSerialized)
        {
```

Each device has a unique identifier, a hardware id (_HID). In this case, it is *IBM0079*, which represents a docking station (GDCK). Most devices provide methods to be called by the operating system, like _DCK for docking, _EJ0 for ejecting or _STA for querying the current status.

The following sections will repeatedly refer to the ACPI specification described above.

3 System as a Whole

The ACPI specification divides states for the system as a whole and for individual components a computer system can contain. Fig. 1 can be used to exemplify how all the states mentioned in the following sections fit together. The whole system is considered a unit which can be put into different operating modes. The highest level of modes are the global states [1, p. 19].

3.1 Global States: G0 - G3

G0: Working System is fully functional, it is in a so called working state. Of course, this is the state with the highest power consumption.

G1: Sleeping System not functional, aka sleeping, but still consuming a low amount of power. However, returning to the working state G0 is possible in a very short time.

G2: Soft-off System not functional, it is off. However, the ATX-Standby power supply is still attached, making it possible for the system to be powered on via remote calls or by pressing the power button. The system still consumes a low amount of power, but less than in state G1. In order to recover from this state, the system needs to be fully booted, which results in a bigger wake up time as if recovering from the sleeping state.

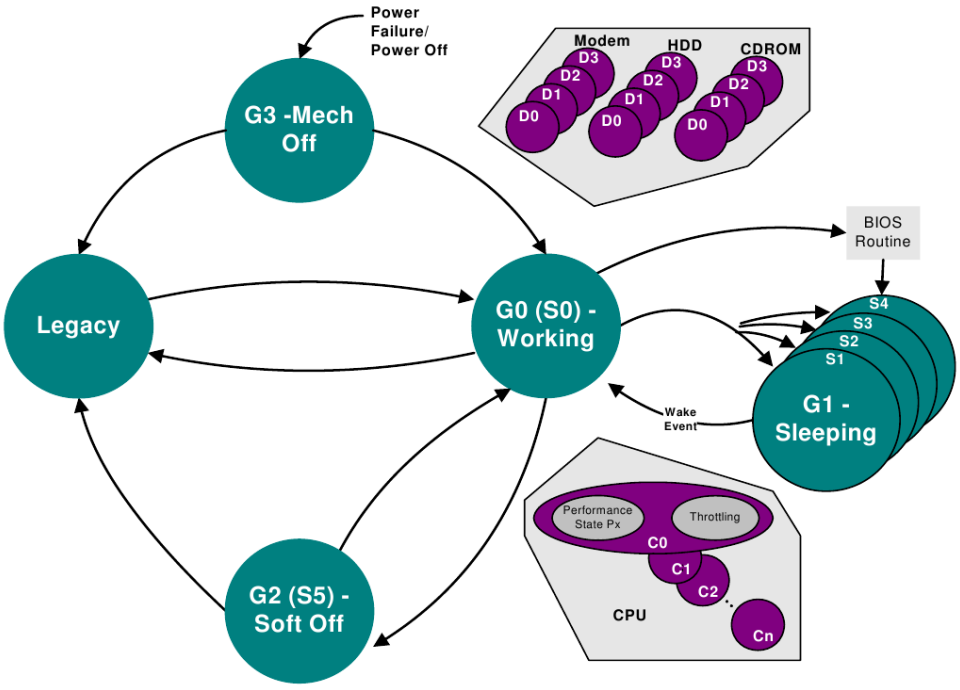

Figure 1: Global System States and Transitions [1]

G3: Mechanical Off The power supply got entirely removed from the system. There is absolutely no power consumption.

All states described here, not only the global states, have something in common. The higher the state, the more time is needed to return to the lowest one. Also, the higher the state, the less power is consumed.

3.2 Sleep States: S0 - S5

The global sleeping state G1 can be subdivided into more fine grained modes [1, p.22]. Starting from sleep state S0, where the system consumes the most power, up to sleep state S5, where the lowest power is drawn. However, one should keep clearly in mind that the higher the sleep state, the more time is needed to return to the working state (cf. G0/S0).

S0 corresponds to the global working state G0. A certain level of redundancy is accepted here.

S1: Standby [1, p. 22] is a low latency sleeping state and some devices are put into a low power mode. However, no system context is lost, so S1 has a very fast recovery time.

S2: Standby is quite similar to S1, except that in S2, the CPU and system cache context is lost.

S3: Suspend [1, p. 22] is widely known as "suspend", "suspend to RAM" or even "standby". It is a sleeping state where the whole system is put into a low power mode. All system context is lost, except the one of the main memory. The active state is saved to fugitive media, the main memory. All other devices are shut down. Due to the main memory still being powered, the system still consumes a low amount of power (about 1 to 3 Watts). On resume from this mode, the active state is read back from the main memory and all devices are powered up again. Usually, setting a system into this state only takes a couple of seconds, the same applies to returning from this state.

S4: Suspend [1, p.22] is also known as "hibernate" or "suspend to disk". In contrary to ACPI S3, the active working state is saved to a permanent media, making it possible for the system and for all devices to be completely powered off. Usually, reaching this state and returning from it lasts about 10 to 60 seconds. S4 shows the same power consumption as with the global states Soft-off (G2) or Mechanical-off (G3).

S5: Soft-Off is similar to G2.

Due to the fact that the operating system is responsible for putting the system into one of these states, it needs more knowledge about system details, like how to power down individual devices or how to put them into a low power mode. This is accomplished through the individual specification parts for most devices shown in the following section.

4 System Components

There are a lot of devices in the system which are covered by general interface definitions. And there are some, such as CPUs, which have their own interfaces, and those which are covered by own specifications, like some PCI devices. This section will cover general device states [1, p. 21], followed by individual component interface definitions.

4.1 Device States: ACPI D0 - D3

Individual devices can be put into a low power mode or powered off through the operating system. That is the reason why the ACPI specification defines special device states [1, p. 21]. A lot of devices and bus systems can be controlled through these D-states. Examples include Card Bus systems, USB devices like mice, keyboards or fingerprint readers, up to firewire.

D0: Fully-On is the active state a device is usually in when it is used. In this state, the most power is consumed, however, there is absolutely no latency when someone tries to access it. In order for the device to be used, it always has to return into this state.

D1 The exact state of each device which is in D1 is defined by each device class. The ACPI specification is kept quite general at this point. It is optional and the only thing that is defined is, that if present, devices in this state should consume less power than in D0.

D2 is equal to D1 when it comes to the the power mode which is again defined by each individual device class. As an optional state, devices may lose some functionality here.

D3: Off means that the device is off. There should not be any power consumption. As a consequence, it brings along a high latency when returning to the working state D0 because the device will require reinitialising.

One has to keep in mind, that a lot of those devices consume power even when they are unused. So the operating system is in charge of putting them into a low power mode (preferable D3) with ACPI in order to reduce the power the whole system consumes.

4.2 CPU

The first important device which is valuable to have an own interface definition is the processor, the CPU. With being one of the biggest power consumption consumers of a modern computer, it also leaves a lot of room for possible power savings. That is why there are even different interfaces defined by the ACPI specification the operating system can access to control the behaviour of a CPU in a fine grained way. The first one are the CPU idle states (C0-Cn) [1, p. 22] explained in the in the next section, followed by the performance states (P0-Pn).

4.2.1 CPU Idle States: C0 - Cn

C0: Working is the usual and only state a CPU can be in when performing operations. Of course, the most power is drawn in this state.

C1 Beginning with C1, the CPU does not perform any operations, thus, is idle. The specification defines the latency when traversing from C1 to C0 to be so small that it can be neglected by the operating system. As a result, in practice, the CPU always finds itself in this state when not doing any work.

C2 - Cn Starting from idle state C2, which includes all the characteristics of C1, the CPU consumes less power bringing along a rising latency when returning to C0. A CPU may or may not define these states. [1, p. 260].

Beside having a possibility to alter the general state of a CPU, either working or sleeping, there is also a more fine grained method to define the performance of the CPU when it is in a working state. This is put into practice with the so called *performance states* discussed in the next section.

4.2.2 CPU Performance States: P0 - Pn

The performance states [1, p. 274] of a CPU alter the clock frequency the processor core can operate with. That is, how many instructions per seconds it can execute. This feature is widely known as *CPU frequency scaling*. Examples for such technologies are *Intel SpeedStep* or *AMD PowerNow*.

Modern operating systems are using this to adjust the frequency on the fly, increasing it when there is heavy work to do, lowering it as soon as there are only tasks which do not need much CPU support. if the OS wants to change the performance state, it issues an ACPI call with the requested state (P0-Pn) and the CPU lowers or increases the core voltage. During this transition phase, the CPU momentarily can not execute any instructions. Each state corresponds to one frequency. For example:

- P0: 1.833 GHz
- P1: 1.333 GHz
- P2: 1.0 GHz

The performance states, and thus the actual frequency and involved latency, may be different for each individual CPU model. It can be read out from an ACPI description table by the operating system.

An aggregation including both CPU idle states and performance states can be seen in the following table for an exemplary Intel Core2Duo Processor [4, p. 77]:

C-State	P-State	Power Consumption in Watt
C1	Pn = 1 GHz	4.8
	P0 = 1.83 GHz	**15.8**
C2	Pn = 1 GHz	4.7
	P0 = 1.83 GHz	15.5
C3	Pn = 1 GHz	3.4
	P0 = 1.83 GHz	10.5
C4		2.2
C5		**1.8**

A first note: There are C-States with operate with no frequency at all (C4,C5).

The one thing that attracts attention is, that there is a huge difference between the CPU being idle in C1 and having the frequency set to 1.83 GHz (P0) and being idle in C5. The power consumption difference is 14 Watt, which is quite noteworthy for a mobile computer.

So as a logical result, operating systems try to balance work as such, that the CPU can be idle as long as possible, saving the most power when having no work to do. In turn, accruing work needs to be processed as fast as possible to be able to put the CPU back to a very low power state immediately afterwards. This strategy is also known as *Race to Idle*.

However, it is not always easy for the operating system to decide which frequency is the best for each workload which might occur. How operating system designers try to circumvent this problem is discussed in the next part.

4.2.3 The Optimal Frequency

Deciding which frequency is the best at each individual point in time is a challenge for the operating system. If it wrongly estimates the occurring workload, bad side effects might be the result. If the frequency is too low, there might be some kind of performance loss. If it is too high in regard to the work which needs to be done, power might be wasted.

There are different criteria an operating system can make use of to estimate the workload and to decide which frequency is the best at a certain point in time. The criteria discussed here is *the process* [5].

Processes that run in an operating system can be separated into two classes:

- **CPU intensive processes**, with a high need for logical and arithmetical operations
- **Memory intensive processes**, which are causing a lot of accesses to the main memory and the MMU

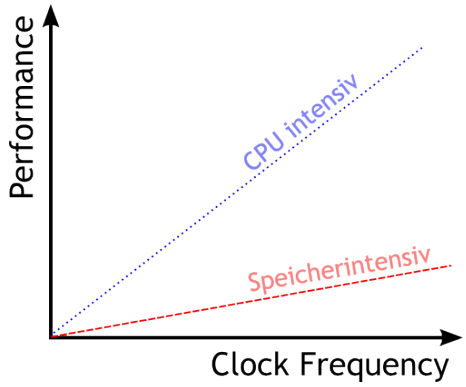

Figure 2: Relationship of performance to clock frequency [5]

As can be seen in fig.2, there is a linear rise of performance for CPU intensive processes if the frequency is raised accordingly. On the other side, this is not the case for memory intensive processes. Increasing the frequency there does not necessarily mean a gain in much performance. That is the reason why the operating system needs decent control and knowledge about the processes which are running, the state and the use case the system operates in.

Of course, for evaluating whether a high, middle, or low frequency is the right choice in a certain situation, the operating system also needs to know how increasing and lowering the frequency is related to the actual power consumption. Fig.3 shows this.

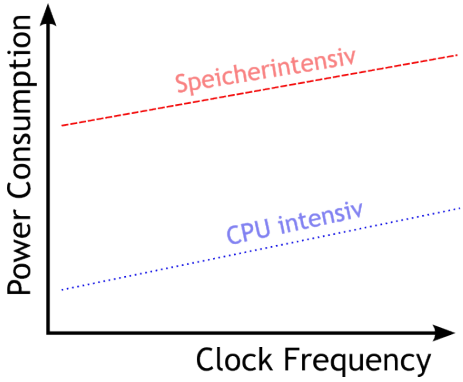

Figure 3: Relationship of energy consumption to clock frequency [5]

Memory intensive processes tend to have a average higher power consumption because more system components are involved.

So, as a conclusion, the optimal clock frequency depends on the used resources. The operating system needs knowledge about the currently running processes and those, which are to be scheduled soon, in order to find and set the best frequency in regard to power consumption *and* performance. One example of solving this are so called *event counters* in the operating system's kernel.

Although CPUs are one of the main power draining part in a computer system, the other components must not be disregarded. One one those other components are displays.

4.3 Displays

Because displays and monitors are one of the mandatory components every personal computer needs, there arises the question about how the power consumption of those can be minimized, or at least reduced to a certain extend.

In general, the power consumption depends on the size, the used technology and the resolution in place. Common CRT monitors need about 50 to 150 Watt, whereas notebook LCDs seldom need more than 3 to 15 Watts. When the computer is in use, this power consumption cannot disappear completely. However, reducing or making it go away when the system is idle should be the major aim. That is where DPMS comes into use.

4.3.1 DPMS

"VESA Display Power Management Signaling (or DPMS) is a standard from the VESA consortium for managing the power supply of video monitors for computers through the graphics card e.g; shut off the monitor after the computer has been unused for some time (idle), to save power." [1]

[1] http://en.wikipedia.org/wiki/VESA_Display_Power_Management_Signaling

Finished in 1993, nearly every modern PC makes use of it. The DPMS specification defines 4 operating modes for a display. On, stand-by, suspend and off. Fulfilling these requirements is not always easy. The display itself, the graphic card and the operating system have to cooperate. The specification needs to be purchased, so here are some indication values a monitor needs to fulfill to be DPMS compliant:

Mode	Power	Recovery Time
On	$< 120W$	0 sec
Standby	$< 110W$	< 3 sec
Suspend	$< 15W$	< 3 sec
Off	$< 5W$	< 20 sec

Beside turning the display off or putting it into a low power mode when not used, there is also the possibility to tweak the power consumption when it is in a working state. Reducing the display brightness.

4.3.2 Display Brightness

Many people actually do not know that reducing the display brightness has the potential to extend the duration a laptop runs on battery to a noteworthy extend. Due to that, operating system vendors use techniques like automatically adjusting the display brightness in certain situations. For example, a lower brightness does make sense when running battery powered, whereas it can be increased when running connected to the main outlet power source. The ACPI specification defines two methods the operating system can use to read out the available brightness levels (_BCL) and to set them (_BCM)[1, p. 584]

Also, the brightness can be dimmed dynamically when there is a short time of user inactivity, not resulting in any functionality loss.

More advanced techniques include ambient light sensors [1, p. 280], which evaluate the ambient light environment the computer is currently used in. The operating system can query those values in order to either reduce the brightness in a dark environment, or to increase it in a bright one. This is considered the most convenient approach, because users do not have to care at all.

As an example about how much power can be saved, here are typical values for a 12.1" LCD display with a resolution of 1024x768 pixels, exposing 7 brightness levels. With highest brightness, it shows an average power consumption of 4 Watt, whereas it only shows about 2 Watt when reducing the brightness to a minimum. This difference of 2 Watt can be quite important on a laptop which has an overall power consumption of only about 10 Watt.

4.4 Hard Drives

Like most other components, the state of storage devices can also be separated into different operating modes. Typically there are three of them, active, standby and sleeping. The operating systems tries to put the hard drive into one of the low power states when

it is not accessed. To extend the time of inactivity, it buffers data as long as possible to prevent the disk to spin up too frequently. However, this brings along the danger of data loss. If there is a system failure for whatever reason, and some data is not yet written to disk, but only buffered, it is lost.

There is actually another risk. The risk of abrasion. Hard disks which come into operation in desktop PCs are often not designed to spin up and down frequently. This might shorten their lifetime. The same danger exists for laptop disks, however, not to the same extend. Manufacturers often construct their mobile disks to be prepared for frequently going to a low power mode and waking up again.

In contrary to that, laptop disks do not contain that much potential of saving a lot of power when in standby or sleep, often they are already optimised for a low power consumption.

Power consumption differs depending on the model. An estimated value of up to 2 Watt can be saved with laptop disks, and up to 15 Watt with desktop disks, depending on the aggressiveness the operating system makes use of the power saving functionality.

4.5 Wireless Networking

One thing, without modern life would be unthinkable, is mobile computing. And another thing which belongs to it is wireless connectivity. So most modern laptops are equipped with wireless network cards making it possible have a live Internet connection while carrying the computer around. Although this is often a mandatory functionality, power consumption factors must not be disregarded. When the wireless LAN standard [3] was designed, the IEEE 802.11 working group already had power management in mind.

The first precondition is to know that the most power of wireless cards is spent during data transmission times. As a logical consequence, the aim should be that there shall only be short heavy burst of transmission, so that connected clients are able to go to sleep immediately after the data was submitted. So clients find themselves in two states. One for transmission of data, the active state, and one where they are sleeping. Clients periodically wake up, checking if there is some data intended for them available.

In turn, the access points are aware of the state of all connected clients, whether they are sleeping or active. If there are data packets for one specific client, and this client is currently in an active state, the packets are immediately delivered. If the client is sleeping, packets are buffered until the client wakes up the next time.

Due to that buffering activity, there is the risk of losing data which would have to be resend in case the access point's buffer is overflowing. Also, there is a certain amount of latency and performance loss in involved. So again, there is a tradeoff between performance and power consumption.

But wireless networking is not the only case where power consumption matters. Wired networks should be considered, too.

4.6 Wired Networking

As most devices, also network cards can be put into a low power mode (cf. ACPI D1-D3). Where this a good possibility when the network is currently not in use, it is a bad idea to deactivate the device if there is data to be submitted or received. So operating systems also have more fine grained possibilities while maintaining full network connectivity. As an example, it could lower the link rate in times of low traffic, e.g. from 100 MBits to 10 MBits.

Another important feature which belongs to power management is "Wake on LAN". This is the possibility to remotely wake up a PC which is currently shut down or sleeping. This is often used in conjunction with the so called "auto suspend". A computer is automatically put into sleep after a fixed period of time of inactivity. To make sure that it can be used whenever someone tries to access it, it can be automatically woken up through "Wake on LAN" from another computer over the network.

The previous sections gave an overview about the mechanisms the most common hardware components provide to do proper power management. Once in a while, there also where short deviations about how operating systems make use of that functionality. The following sections address the strategies, the so called *policies*, operating systems can make use of to do intelligent power management in a general manner.

5 Software Policies

Software policies can be separated into two classes [2, p. 10]. Static ones, which are defined at software design time, like designing a resource efficient and intelligent algorithm. These policies cannot be altered later on.

In contrast to that, dynamic policies define and change the behaviour at runtime. Those dynamic strategies try to take advantage of the power savings potential when there is only low or no demand for functionality of certain hardware components. This includes putting them into a low power mode (ACPI D1-D3) or adjusting their performance (cf. CPU frequency scaling). A generic term for this is *Dynamic Power Management (DPM)*.

At a first glance, this seems quite simple. Why not just putting the device to sleep as soon as there is no work to do and waking it up as soon as there is? The reason why that does not work is that there are some drawbacks involved when putting a device into a low power mode and when recovering from it.

Fig. 5 illustrates a typical work flow of a device. The top row indicates when the device is accessed. Whenever it is unused, it is idle (T_1 to T_4). If the device is idle, it can be put into a low power mode or can be shut down.

So as a first problem, there is a delay involved when the device changes its state to (T_{sd}) or from (T_{wu}) a low power mode. During this transition time, the device cannot be used for a short moment. For example, a computer user finds it quite annoying if he frequently tries to listen to music songs and always has to wait two seconds for the song to begin playing.

Also, when transitioning from one state to another, either waking up or going to sleep,

most devices exhibit a short energy peak as can be seen in fig. 4.

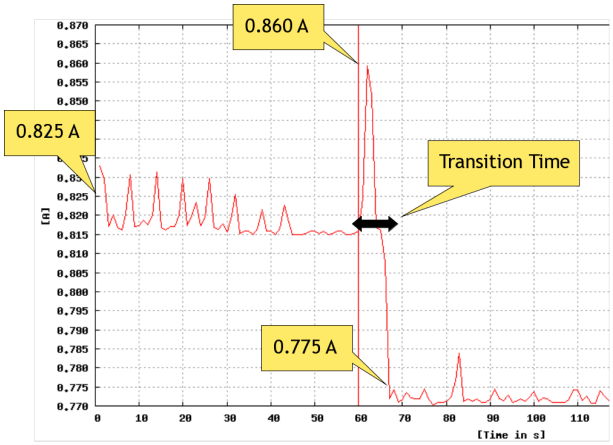

Figure 4: Exemplary transition phase

The challenge of the power management policy is to find a way to compensate these problems while still saving as much power as possible. So two major aspects need to be considered:

1. The total power savings while sleeping must be higher than the increased energy needed for putting the device to sleep and waking it up.
2. Too frequent sleep interruptions after only a short sleep time must be avoided to not hinder the user's method of operation.

For archiving this goal, different policy models come into question.

5.1 Time-Out

Many people might already have experienced that their display will go off when they are not actively using it (cf. DPMS) or that their hard disks spin down from time to time. Those are perfect examples for a time-out based policy [2, p. 12]. That is, after a specific period of idle time τ (T_1 to T_2, cf. fig. 5), a device will be put into a low power mode or will be turned off.

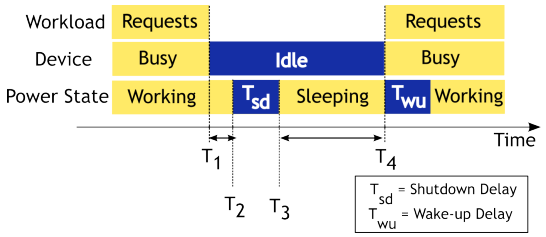

Figure 5: Time-Out Policy [2]

The time-out policy presumes, that if a device is idle for a specific period of time τ, the device will be idle for another period of time T_{be} (T_2 to T_4). The power consumption wastage during the time-out period τ where the device is idle but still not powered down is accepted.

Even the time-out policy model can be differentiated further. There might be a static timeout value τ, for example something like 15 minutes until the display gets turned off. Or there can be a dynamic time-out automatically adapted to the current situation and requirements. If the policy notices that there are too many prematurely sleep interruptions, it could increase the time-out value, respectively it may lower it if the device is not accessed too frequently. Also hardware characteristics could be taken into account.

5.2 Predictive

Another approach for a proper power management policy is called *predictive* [2, p. 12,13]. This policy tries to forecast the duration of the *next* idle period. This can be archived with different techniques. One example would be to gear to the duration of previous idle periods. Another would be to take typical user behaviour into account.

Break-Even Time To fully understand fig. 6, the term *Break-Even-Time* needs to be made clear. It is the span of time from T_1 to the break even point shown in fig. 6. The Break-Even Time is the period of time that a device needs to be idle so that it is worthwhile for it to be put into a low power mode. This period is calculated of the sleep and wakeup delay (T_{sd} and T_{wu}) and the energy raise resulting from those transitions.

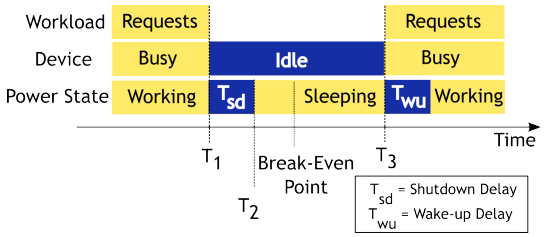

Figure 6: Predictive Policy [2]

So if the predicted idle period of a device is greater than the break even time, the device is immediately put to sleep (T_1), eliminating the power wastage of the time-out period τ mentioned in the previous section. However, and important to mention, is that the overall power savings depend on the accuracy of the forecasts.

5.3 Stochastic

Also stochastic methods [2, p. 13] could be used. The decision if and at which time a device is put into a low power mode is based on stochastic probabilities. Requests may arrive at a certain time and a device sleeps at a certain time. The goal is to solve the

stochastic optimization problem of these two probabilities. If this is done, the policy exactly knows when it makes sense to put a device to sleep. If it knows that it will be very unlikely that there will be requests until the break-even time has elapsed, it can immediately put the device to sleep.

5.4 Recapitulation

As a conclusion, the policy in place, and if it makes sense, heavily depends on the use case. User behaviour is unpredictable and power savings most often bring along some kind of performance loss. The higher the savings, the worse the performance. The higher the performance and responsiveness, the worse the power savings. So a general goal has to be to keep output losses as low as possible while still getting the most possible power savings. There is always a tradeoff between performance and power consumption.

6 Conclusion

Having ACPI as the basis for power management in the computer area, it is important to mention that due to the fact that the hardware and BIOS do not interfere, or only to a very low extend, the benefits heavily depend on the current implementation. It is the duty of the operating system designers to make use of all hardware features and to integrate them in a reasonable way. Also, application developers need to be aware that power management needs to be considered in every piece of software. In the end, there will only be noteworthy achievements if all components, from the applications, over the operating system, down to electronic circuits, play together.

And this is the challenging part everybody has to face. Applications are made by certain companies, operating systems are made by others, same for the hardware. Not to forget the users, which play a remarkable role when it comes down to how the hardware and software is used. A good cooperation between all involved parties is needed to become successful and for the efforts to pay off. The time is over to blame other people. Everybody has to point the finger on himself.

References

[1] *Advanced Configuration and Power Interface Specification*, October 2006. http://www.acpi.info.

[2] Luca Benini and Giovanni de Micheli. System-level power optimization: techniques and tools. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 5(2):115–192, April 2000.

[3] IEEE Standards Association. *IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements*
Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, January 2008.

[4] Intel Corporation. *Intel® Core™2 Duo Processors and Intel® Core™2 Extreme Processors for Platforms Based on Mobile Intel® 965 Express Chipset Family*, January 2008.

[5] Andreas Weißel and Frank Bellosa. Process Cruise Control-Event-Driven Clock Scaling for Dynamic Power Management. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES 2002)*, Grenoble, France, October 2002.