解　説

ノートブックPCのパワーマネージメント

Power Management for Notebook PC by Susumu SHIMOTONO(Project Manager of Low-Power Systems Institute, Tokyo Research Laboratory).

下　遠　野　享[1]

1 日本アイ・ビー・エム(株)東京基礎研究所

1. はじめに

ノートブックPCが1990年頃からモノクロLCDパネルとニッカド(NiCd)バッテリーという組合せで市場に発表され始めて，パワーマネージメント(以下PMと略す)と呼ばれる省電力技術の必要性が注目されるようになった．注目された理由は簡単である．この当時のノートブックPCはその内蔵バッテリーで最大2時間程度しか連続動作できなかったからである．この欠点をどのように解決するかがPM技術の課題であった．そして現在この連続動作時間への要求はますます高まっている．実際の使い方で1つのバッテリーによる連続動作が丸1日(たとえば8時間)を保証するノートブックPCは現状では存在しない．これまでのPM技術の進歩で実現してきた消費電力低下への努力やバッテリーの進化の成果の多くが，昨今の急速なPCの高速化や高機能化にともなう電力消費の増加で相殺されているともいえる．今後予想されるより高い携帯性（軽量化）への要求もこの技術の重要度を増大させている．

本稿では上記課題の解決のためにこれまで採用されてきた具体的な方法を述べながらPMの機能を解説するとともに，それらの方法を実装するまでの技術的な発展の経緯にもいくつか言及する．最後に今後の動向についても述べる．

2. PMの課題の解決のために

上記の欠点解決のために，いろいろな新機能や改良が考えられ，それらの多くが実際に実装され，今現在もそれらの性能向上は続いている．それらを列挙すると，

(1)　バッテリーで使用中にその状態を保持したままバッテリー交換のため一時中断して，その交換が完了すると元の状態に復帰して使用を再開できる機能を提供する．

(2)　充電した全電力量の最大限の利用のための正確な残量表示機能とバッテリー残量ゼロ間際の緊急避難的な対応機能を提供する．

(3)　より大きなバッテリーが搭載できるように内部の各種コンポーネントの実装密度を向上させたり，筐体そのものを大きくしたり，複数のバッテリーが搭載可能なように2次記憶デバイス(FDDなど)のスペースを第2のバッテリーと共用にしたりする．

(4)　アプリケーションプログラムにおけるアイドル状態をソフトウェア技術で検出して，その時のCPUおよび周辺回路の動作クロックを低速にしたり場合によっては停止する．

(5)　システムの中で活動中でないデバイスやサブシステム(HDDやLCDなど)を検出して，それを低消費電力状態に遷移させる．

(6)　高速クロックで駆動される回路を最小化したり，低電圧化(5Vから3.3Vなど)をはかるなどにより，デバイスやサブシステム自身の消費電力を低下する．

(7)　バッテリーの高性能化やバッテリーからの供給電圧を変圧し安定化させるDC/DCコンバータの高能率化をはかる．

上記(1)および(2)はバッテリー駆動の多くのPCで広くサポートされている．これらの機能はバッテリーでの動作時間の延長を狙ったものではないが，もしこれらの機能がなければユーザーはバッテリーでノートブックPCを使用中に重要で込み入った仕事を安心して行えない．使用の真っ最中に不意にバッテリー残量がないと知らされて

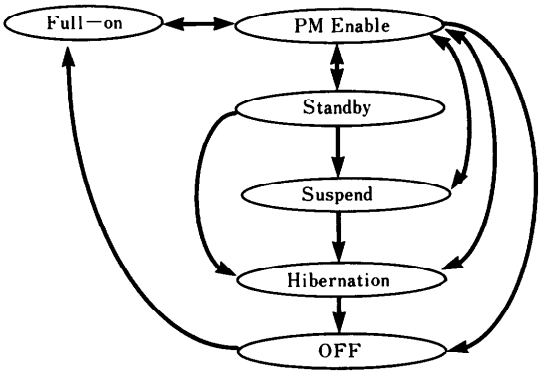


図-1 PM状態遷移

も，うまくそれに対処できるユーザーは少ないからである．つまり，いわば保険のようなものであるが，これらの機能はノートブックPCでは必須である．これらについては後述する．(3)はバッテリー動作時間の拡大には最も即効性があり製品開発では第1に配慮されていることである．しかし本稿の主旨とは異なるので本稿では言及しない．(4)および(5)はシステムの電力の利用効率の向上を狙うものである．これらについても後述する．(6)はPMに限らず性能向上(処理速度)と発熱の問題として常に議論されることである．この分野の進化はノートブックPCのバッテリー動作時間延長に直接大きく貢献している．さらにシステム全体の低消費電力化だけでなく，発熱を抑えることでシステムの高密度実装を可能にし，転じて筐体内のバッテリー用スペースを拡大させることで間接的にも貢献している．ただし，一般的な議題なので本稿では割愛する．(7)は昨今のバッテリーの進化(リチウムイオン電池の普及)など特筆すべき話題がある．ただ，バッテリー自身の性質に関することなのでこれも本稿では割愛する．

3. PMの必須機能

前述したバッテリー交換のために必須となる一時中断機能の例のようにシステムはパワーの観点でいくつかの状態をもつ．図-1はその状態遷移の例を示したものである．所定のボタンを押したり，LCDパネルの開閉などが，その遷移のきっかけになる．この章では，その各状態の概要とその実装の例を述べる．またもう1つの必須機能である高精度の残量レベル管理が可能なインテリジェント・バッテリーについても述べる．

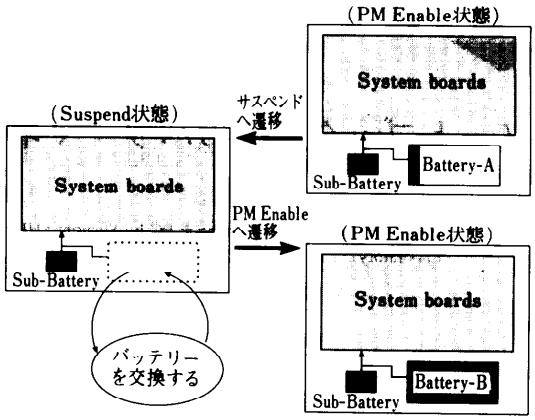


図-2 バッテリー交換

3.1 低消費電力状態の概要

3.1.1 システムサスペンド

前章で述べた一時中断機能は通常サスペンド・レジューム機能を指す場合が多い．この機能は日本では単にレジューム機能として言及されることもある．この中断(サスペンド)状態ではその状態を維持するために最低限必要なデータのみが保持される．それらはシステムのメモリの全内容とこの状態でOFFされるデバイスやCPUのレジスタの全内容である．(各デバイスとCPUのレジスタは読み出されてメモリ上で保持される．)それらのデータ維持のための最低限のパワーのみが供給される．通常(PM Enable)状態へ復帰(レジューム)すると，それらの保持データがすべて元の状態に回復され遷移直前のプログラム実行アドレスの直後から再開する．OSの管理するシステム時刻もある手順で同時に回復されるのでPMを管理する一部のソフトウェアを除いてどのプログラムも(OSの大半も)中断されていたことに気づかない．

サスペンド中の消費電力は動作時に比べると実に1/100前後に激減する．これによって，たとえ使用中にバッテリー残量がゼロに近くなってもサスペンド状態に遷移すれば外部電力の補給までに数時間の猶予が与えられる．また通常の着脱式バッテリー以外に，上記1/100程度の電力消費なら数分程度システムを維持できる随時充電可能で小型の固定式バッテリーもあわせて内蔵しており，これがサスペンド中のバッテリー交換を可能にする．図-2にその交換の様子を示す．この機能を前提にバッテリー残量がゼロ近くになると自動的

にサスペンド状態に遷移する機能をサポートするのが一般的である．これが前章で述べた緊急避難的対応にあたる．サスペンド・レジュームはその遷移が数秒程度ですむため，このような一時中断には非常に便利な機能である．

3.1.2 システムハイバネーション

サスペンド・レジューム機能とよく似た機能としてハイバネーションと呼ばれる機能がある．ハイバネーション(Hibernation)が冬眠という意味であるように，この状態ではバッテリー電力をまったく消費しないので実質的には永久にその中断状態を(ハードディスクに)保存し続けることができる．サスペンド状態で電力消費していたメモリの内容などをハードディスクに写しとることによって電力を消費しなくてもすむようにしたわけである．

サスペンドのように高速で遷移することはできないが(メモリサイズによってハードディスクへのデータ転送のため，十秒から数十秒程度の遷移時間を必要とする)，この電力をまったく消費しないという特長はサスペンドの機能を補完する有効な機能となっている．たとえば，サスペンド状態のままある一定のバッテリー残量以下になった時，自動的にハイバネーション状態に遷移する機能がある．前述のようにサスペンドでも微少ながら電力は消費され続けるので，サスペンド状態に入ることで保持した(つもりの)データをうっかりそのまま長時間放置して消失することが少なくなかった．このハイバネーション機能の出現によって持続時間に制限があるサスペンド・レジューム機能をその長所(高速遷移)にのみ焦点をあてて活用できるようになった．

3.2 サスペンド機能実装の特徴

図-3 は代表的なハードウェアのブロック図の例をそのパワー供給の観点でまとめたものである．簡単にいうと，サスペンド機能のサポートのためデバイスのパワー供給系が2分割されている．サスペンド時に点線で囲まれている部分以外のパワー供給は遮断される．メインメモリやVRAMはサスペンド中，スローリフレッシュやセルフリフレッシュで低消費電力化される．また，ほかのONのデバイスも非同期の動作要求をされないので静的に低消費電力状態になっている．

このようにサスペンド時には2分割されたパワー供給系の一方を遮断し，他方のパワー供給で維持されているデバイス群も特別な低電力状態になっている．また通常は互いに接続されて動作しているこれら2つのデバイス群がサスペンド時には電気的に分割され特殊な状態になるので，ONのデバイスとOFFのデバイス間の信号線を介して不要な電流が流れ出さないように絶縁回路を設けたり，OFFデバイスに接続されたONデバイスの出力信号を状態遷移直前に意図的にLOWステートにするなどの配慮もされている．

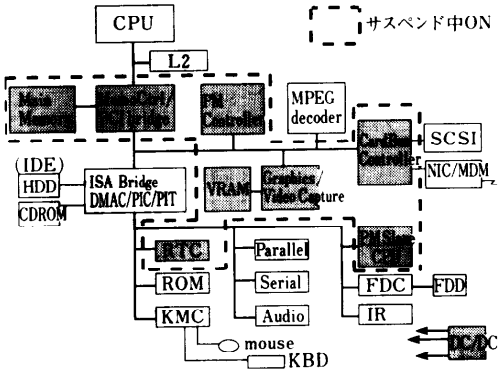


図-3 2系統の電源

3.3 ハイバネーション機能実装の特徴

ハイバネーション機能の実現のためには最大でメモリサイズにほぼ等しい空きエリアをハードディスクなどの2次記憶装置内に予約しておく必要がある．焦点となるのはその空きエリアの確保の仕方である．1つの方法はディスクパーティション(Disk Partition)を使うやり方である．本来，ハードディスクは異なるファイルシステム(つまり，場合によっては異なるOS)が1つの物理ディスクに共存できるように，複数のディスクパーティションに分割可能である．空きエリアの確保として最も単純なのがハイバネーション専用のディスクパーティションをOSのインストールの前にあらかじめ1つ確保してしまうことである．ただし，システムメモリの増設対応のためにはそのマシンがサポートできる最大搭載メモリサイズに等しいパーティションをあらかじめ確保する必要があるという欠点がある．ディスクパーティションサイズはあとで変更できないからである．もう1つの方法としては，PMコード自身がサポートすべき特定のOSのファイルシステムを理解し，ハイバネーション用の空きエリアはそのファイル

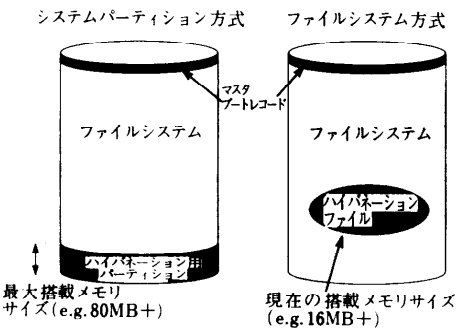


図-4 ハイバネーションのためのハードディスク使用方法

システムに1つの巨大な隠しファイルとして確保するという方法がある．実装方式としてはOSのファイルシステムに依存するなどの欠点はあるものの，ユーザにとってはこの方が使い勝手はよい．**図-4**にこの2つの方式の概略を示す．

さて，ここで注目すべきは，PMコードがハイバネーション機能でハードディスクをアクセスする方法とその環境である．それらは通常のプログラムがファイルハンドルを使ってOSに依頼する場合とは大きく異なる．なぜなら，ハイバネーションの遷移中はPMコード自身が管理するメモリ領域を除く全メモリの内容への変更を，すべてのコード(OS／アプリケーションソフトおよびPMコード自身)に対して禁止する必要があるからである．その禁止の理由を示す例を1つ述べる．

ハイバネーションへの遷移中(ディスクにメモリがセーブ(保存)されている真っ最中)に，あるプログラムの動作にともなってメモリの複数個所が変更されるとする．その内の一部(変更Aとする)はPMコードがすでにディスクにセーブずみの領域，他方(変更B)はまだこれからセーブされようとする領域だとする．ハイバネーションからPM Enable状態に復帰して通常動作に戻り上記のハイバネーションへの遷移中に動作中だったプログラムの実行も再開されると，そのプログラムがメモリにセーブしたはずの変更Aのデータが回復されていないことになる．たとえばその失われたデータがあるプログラムへの戻りアドレスだったりすると回復不能のエラーとなるし，管理データだとするとその後の動作が不調になる．たとえていうなら，シャッタースピードが十秒から数十秒(ハイバネーションの遷移時間)かかるカメラで撮影する際，被写体が動くと(メモリ書き込みの発生)きちんとした画像が得られず，その時点の状況を再現(ハイバネーションからの復帰)できないわけである．

したがってPMコードはハイバネーションのためにほかのいっさいのプログラムが動作できない特別な状況を作り，独自のディスクアクセスプログラムによってメモリの保存・回復を実行する必要がある．

3.4 状態遷移時に考慮すべき点

初期(1990年頃)のPMはOSとの協調によって実行されていたわけではなく，逆にOSのまったく知らないところでPMコードを独自にROMに実装して多くの機能を実現していた．そのような独自実装方式だったため，(1)低電力状態からの復帰時に，IBM PC/AT互換のI/Oポート[1]でいくつかの書き込み専用ポートのデータを回復できなかったり，(2)動作の真っ最中だったデバイスを強制的にOFFしたため，復帰時に同じ動作状態に回復できなかったり，(3)OSがBIOS[2](バイオスと呼び，システムごとの差異をOSやアプリケーションの階層から隠すシステムプログラムでここではPMコード)に通常与えるシステム保護用特権レベルでは特定のI/OポートやCPUの内部レジスタにアクセスできなかったりするなど，複雑な問題が発生した．

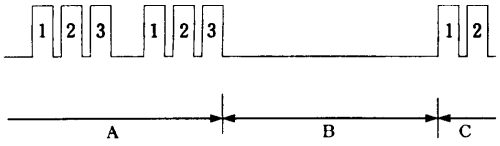
これらを解決するために次に述べるような対応がはかられた．(これらの対応方法はすでにデファクト・スタンダード化し，皮肉なことにPMに対してOSの協力がある程度得られるようになった現在でも，逆にOS側がそれを前提にしてPM機能の拡張をはかっている．)

3.4.1 書き込み専用I/Oアドレス

問題の書き込み専用レジスタと同じ書き込みアドレスをもち，その読み出しアドレスはIBM PC/AT互換のI/Oアドレス範囲外にもつシャドウ・レジスタ(Shadow Register)と呼ばれる隠しI/Oポートを設けそれら書き込み専用レジスタに対応した．

3.4.2 デバイス動作中の状態遷移

動作中のデバイスに絡む問題の一例として，**図-5**にPS/2マウスのシステム本体へのデータ転送の様子を簡単に表す．マウスはそのボタン状態(第1バイト)および移動量(x-y軸として各々第2および第3バイト)の3バイトを1つのパケッ



A：マウスが連続動作中
B：マウスの動きが一時的に停止
C：マウスが連続動作再開

図-5 マウスの基本動作

トにして送出する．しかしこのパケットデータには先頭バイトであることを示す情報がない．先頭バイトの区別がつかないため，マウスデバイスドライバはその送出されてくるバイトデータの順番だけを頼りにその内容を解釈している．したがって，マウスが連続的に動かされている(デバイスBUSY)時に省電力状態への遷移が発生すると，PMコードはマウスからのデータがパケット単位で到着するその区切りのタイミングでその送出を停止させる命令を送った後マウス電源のOFFを行う必要がある．さもなければ，マウスは不完全なデータパケットを送信したままOFFにされ，PM Enable状態への復帰(つまりマウスの電源ON)後再び通常のデータパケットを送信し始めるため，マウスデバイスドライバはバイトデータの順番を取り違えたままとなり，画面上のマウスポインタは不自然な動きをし続けることになる．

このようなデバイスBUSYと状態遷移(PMの処理)の衝突での唯一の解決策はPMの処理を僅かな時間だけ延期することである．たとえば，PMコードはこのようなデバイスBUSYの状況を1つでも検出すると自己の割り込みを数十msec後に再度発生させるようにPM関連のハードウェアをセットして終了し次の機会をうかがうような振舞いをする．これを所定のタイムアウト時間まで繰り返す．つまり，瞬間的にデバイスが皆アイドルになるチャンスをみつけるのである．サスペンドへの遷移時(システムがOFFする直前)に，マウスを意図的に動かし続けるとなかなかサスペンドへ遷移しないのはそのためである．

もしマウスデバイスドライバにこの状態遷移が通知されてデバイスBUSYとの衝突に適切に対応できれば，PMコードがデバイスBUSYに対してこのような配慮をする必要はないはずである．この発想でAPM[3] (Advanced Power Management)と呼ばれるOSとROM(PM)の間の取り決めが1992年1月にIntel社とMicrosoft社からAPM 1.0として発表され，メーカがシステムに新しく追加するデバイスとPMとの関係においては一応の成功を収めている．しかし，このAPMはOSの標準デバイスドライバには波及されず，依然として現在もPMコードはこの例で述べたような配慮を多くの標準デバイスに対してする必要がある．(書き込み専用レジスタに対しても同様である．)前述したように，状態遷移時にAPM出現以前に確立されていたPMコードの独力での機能を，OSが暗黙に前提にして動作しているからである．

3.4.3 PMコードの特権レベル

これはWindows3.0になってBIOSをDOSのエミュレーションモードで動作させるようになってから問題になってきたことである．1991年，Intel社はSMM[4] (System Management Mode)と呼ぶ内部マクロを追加した新しいx86CPUアーキテクチャを発表した．このSMM内で特徴的なことは，PM用に新しい外部割り込みとしてSMI(System Management Interrupt)を備えPMに関する処理をほとんどすべてこのSMIハンドラー内で実行できるようにしたことである．このSMI内ではPMコードが最高位で唯一の特権レベルをもつことができる．さらに，ほかのメモリ領域とはまったく切り離された特別のメモリバンクをPMのコードおよびデータ専用にハードウェア的に提供できる．これで各社は特権レベルの問題から解放され，さらにPMをOSのタイプやバージョンによらず独立して開発できるようになった．現在すべてのx86CPUベースのPCのPMコードはSMIで実行されているといっても過言ではない．

3.5 インテリジェント・バッテリー

ここではシステムとバッテリーの関係について述べる．1992年頃からバッテリー内部にCPU(4ビットクラスの軽量なもの)を内蔵した，いわゆるインテリジェント・バッテリーが注目されだした．この内蔵CPUはシステム本体と低速の非同期シリアル通信などを行うことにより，きめ細かなバッテリー制御やその状態の把握を可能にした．

それまでは，実際の使用時のバッテリーの出力電圧レベルなどを主に使用して残存電力量を推測

していた．しかしその推測値の精度には限界があった．そこでCPU内蔵のバッテリーが出現し，そのバッテリーの入出力電力量を内部温度なども勘案しながら逐次監視して最新の残存電力量を個々に内部保持するという考え方が導入された．

これで精度が飛躍的に向上しシステムは定期的にその残存電力量をバッテリー自身から受け取ることによってシステム全体の単位時間あたりの消費電力を動的に計算できるようになった．そしてその時点での残存電力量との組合せから現在の電力消費ベースであとどのくらいバッテリーがもつのかを時間に換算してユーザに随時報告することも可能になった．

4. 電力消費の効率化

4.1 CPUアイドル

昨今，CPUの消費電力は周辺のL2キャッシュやホスト(CPU)バスブリッジなども含めると最大で5W前後まで増大しシステム全体の3分の1を占める場合もある．この電力消費を効率化することは低消費電力化への有力な方法の1つである．通常のビジネスパッケージソフトはほとんどがインタラクティブな(ユーザーとその仕事を処理するソフトとが相互に作用しあうような)ものである．そのようなソフトはユーザーがキーボードやマウスを操作しなくなると単にユーザーの次の操作を待っているだけとなる．この時，例外はあるが(後述する)，OS内のタスク・スケジューラは定期的で非常に短いタイマ割り込みの処理タスクを終了すると毎回実行すべきタスクが一時的にない状況になる．これをCPUアイドルと呼ぶ．

4.2 APMアイドル通知

前述したAPMの仕様にはOSがCPUアイドルを検出するとAPMのインターフェースを経由してAPMBIOS(PMコードに相当)にその通知をする機能も盛り込まれている．これをサポートしたOSとしてDOS5，OS/2，Windows3.1，そしてWindows95がある．その通知を受けて，通常次の外部(システムタイマなど)割り込みが発生するまでPMコード内部でCPUクロックは停止する．タイマ割り込みに要する処理時間は通常1%にも満たないので，CPUアイドルが続く場合実にCPUクロックは99%以上停止していることになる．

Intel社のx86系のCPUはストップクロック(STPCLK)と呼ばれる入力信号の操作で，CPU内部のPLL回路およびキャッシュとメモリとの同一性維持のためのバス監視用回路を除くそのほかすべてのクロックを停止し，1W弱まで消費電力低下が可能である．さらに条件が揃えば，CPU全体のクロック停止も行い，ほぼ消費電力ゼロになる．これらの状態から通常動作状態への復帰には単に外部割り込みが発生するだけでよい．

4.3 APMアイドル通知の限界

インタラクティブなアプリケーションソフトが動作中であれば必ずCPUアイドルの検出を経てCPUクロックが停止するわけではない．そのアプリケーションソフトがユーザー入力を待つ間OSにその実行権(CPU資源)を解放するか否かでCPUクロックが停止できるかが決まる．

ところが，PCの主流ビジネスソフトは以前からのバージョンアップ版が多く，インタラクティブな処理とそのインタラクティブな処理内容をきっかけに発生するバックグラウンド処理を同一プログラムループに入れている例が多い．バックグラウンド処理中にもインタラクティブ処理をサポートできる構成にすると，インタラクティブ処理が終了しても安直にOSにその実行権を解放しないようにする必要が出てくるわけである．これがインタラクティブアプリケーションの動作中でもシステムがアイドルになかなかなれない原因の1つとなっている．

今後，32bitマルチスレッドのプログラミングが次第に主流になるに従い，別スレッドに切り出されたバックグランド処理の起動がOSのスレッド間通信でオンデマンド(On-Demand)で行われれば，そのアプリケーションがCPU資源をつかんだまま放さないということは少なくなる．これによってインタラクティブ処理時のCPUおよびその周辺の消費電力は安定的に低下する．周辺ハードウェアの低消費電力化とCPUのパフォーマンスアップによるその消費電力増でCPUの消費電力のシステム全体に占める割合がますます増加するなかで，これは非常に注目される分野である．

4.4 CPUクロックのスピード調整

インタラクティブ処理中のCPUアイドルが正確に検出できないとすれば，そのような時に平均のスループットを低下させずに消費電力を低下さ

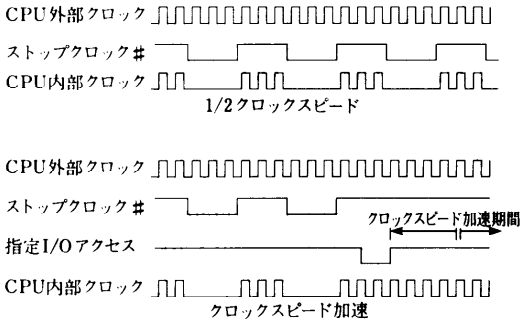


図-6 CPUクロックのスピード調整

せる方法として，CPUのクロックスピードを低下させることが1つの有効な方法となる．ただし，CPU内のPLLで作成されている内部クロックはその周波数を遅延なく動的に変更することができない．そのため周期的に前述のストップクロックで内部を停止してCPUを間欠動作させ疑似的に低速のクロックスピードを作る．そして，I/Oアクセスの頻度とシステムのアイドル度は反比例の関係にある場合が多いので，通常その低速クロックで動作していて，あらかじめ指定したI/Oアクセスが検出されるとストップクロック信号の操作をある指定時間停止してCPUの処理能力をアップさせる方法も採用されている．図-6はその可変速CPUクロックの概観図である．参考のため，ストップクロックの周波数を高くしてある．

4.5 デバイスアイドル

ある期間未使用の場合低消費電力状態にされるものとして，HDD，CD-ROM，LCDパネル，モデム，オーディオなどがある．そして昨今PCIバス自身の低消費電力化が視野に入ってきた．

PCIの規格が2.1になってPCIクロックをバスが未使用であればいつでも停止する機能が追加され今後PCIバスに接続されるデバイスの消費電力を場合によっては飛躍的に低下できる可能性がある．

5. プラグ アンド プレイ（PnP）

Microsoft社が1995年8月にWindows95を発売する際に発表したPC95[5]と呼ばれたリファレンスモデルのなかに“プラグ アンド プレイ(以下PnPと記す)[6]”のコンセプトがあった．この機能は一言でいえば，デバイスのシステムへの設定(コンフィギュレーション)を自動化する機能である．この自動化の実現によりデバイスによってはパワーON状態での抜き差し(ホットプラグ)が可能になった．この考えはPMにとっても非常に意義深いものとなった．なぜなら，PMとPnPはそのデバイスに対する処理が究極的には同じになるからである．たとえば，あるデバイスをホットプラグアウトする(つまり，動的にそのデバイスを引き抜く)とすると，それは消費電力をセーブするためにPMがそのデバイスのパワーを動的にOFFすることと同じことになるからである．反対に，ホットプラグイン(デバイスを差し込む)なら，PMによるそのデバイスのパワーON時の初期化処理などと同じになる．

このようにこのコンセプトはPMの今後に新しい流れを提起した．ただし，Windows95ではPnPをもってPMの多くの機能を代替するとは宣言していない．現在のPnPの主目的はシステムへのデバイスの自動設定である．PMの動作ポリシーも依然マシン固有のユーティリティーに依存している．OSがPMを統合的に管理しているという状況ではない．DOSやWindows3.Xのコードもほぼ正常動作可能な下位互換性の保証という命題のためPMとの関係も継続したものと思われる．

6. 今後の動向

Windows95のためのPC95に続いて，1996年3月，Microsoft社は次のOSメジャーバージョンアップ用にPC97と呼ぶリファレンスモデルを発表した．この中で特徴的なのが“OnNow[7]”というコンセプトである．簡単にいえば消費電力を極力低くしたOFFのようにみえる(テレビの待機モードのような)ON状態をPCで実現しようとするものである．これまでPCといえば常にON状態を前提にすべて(OS／アプリケーション／デバイスドライバ／そしてハードウェア)が組み立てられてきた．今後PCがホームPCとして大規模に展開されようとするとき，この前提が一般家庭では受け容れ難いものだというのが“OnNow”の主張である．つまり，もっと家電製品のように簡単にON/OFF(のようにみえるON)が可能になるべきだということである．この“OnNow”実現のためにMicrosoft社は次期OS(Windows OS)でPMをいよいよ本格的に本

体に取り込むことを宣言している．Windows95とPC95でPnPを業界標準にしたMicrosoft社の隠然たる市場支配力を考えると，このOnNowがユーザーからみたPCのイメージを一変させるかもしれない．ここでは，ついにPMとPnPは融合され，さらに，BIOSやSMIなどのx86CPU依存からの脱却も同時に試みられようとしている(ACPI[8] (Advanced Configuration and Power Interface)の導入)．アプリケーションのためのPM用APIも積極的に用意すると宣言されている．昨今デスクトップPCでも熱問題などで否応なくPMに関心をもつようになっており，今後それらのAPIを使ってPMを強く意識したアプリケーションがそのバージョンアップのタイミングで発表されるだろう．また，OFF(のようにみえる)状態でもネットワーク接続可能な機能も実現可能となるだろう．

7. おわりに

ノートブックPCにほぼ限定されてきたPMもメジャーOSのフルサポートによる“OnNow”の導入で数年後にはいよいよPCの檜舞台に登場する時期がやってきたようだ．PCのイメージを大きく変える主役として躍り出るかもしれない．デスクトップPCへの展開で，“PM＝ノートブック”の観念も次第に薄れてゆこう．

参考文献

1) IBM: Personal System/2 Hardware Interface Technical Reference(Common Interfaces).
IBM: Personal System/2 Hardware Interface Technical Reference(System Specific Information).
2) IBM: Personal System/2 and Personal Computer BIOS Interface Technical Reference.
3) Intel Corporation, Microsoft Corporation: Advanced Power Management(APM) BIOS Interface Specification Revision1.0 (Jan.1992).
4) Intel: Intel486SL Microprocessor SuperSet Programmer's Reference Manual (1992).
5) Microsoft PRESS: Hardware Design Guide for Microsoft Windows95.
6) Compaq Computer Corporation, Phoenix Technologies Ltd., Intel Corporation: Plug and Play BIOS Specification Version1.OA (May 5, 1994).
7) http://www.microsoft.com/developer/tech/hwdev/onnow.htm
8) Intel, Microsoft, Toshiba: Advanced Configuration and Power Interface Specification Draft Revision0.7 (June 1996).

(平成8年9月10日受付)

下遠野 享
1957年生．1982年中央大学理工学部電気工学科卒業．同年(株)リコー入社．1989年日本アイ・ビー・エム(株)入社．以来，ノートブックPC(ThinkPad)，RS/6000ノートブック(ThinkPad Power Series)のパワーマネージメント(とくにパワーマネージメント用ソフトウェア)の開発設計に従事．1996年現所属に異動．パワーマネージメントの研究開発に従事．