《研究報告》

Windowsコンピュータを使ったサブミリ秒計時手法の比較

齊藤恵一　河内哲也*

A Comparision of Submillisecond-Timing Methods on Windows PC.

Keiichi Saito　Tetsuya Kochi*

Abstract : Accuracy and reliability of three submillisecond-timing methods, a function *QueryPerformanceCounter()*, an instruction *RDTSC*, and a method *IReferenceClock::GetTime()* was evaluated under the Microsoft Windows operating system. Measuring frequencies of voltage alternation produced by the three timing methods revealed that the *QueryPerformanceCounter()* function and the *RDTSC* instruction performed at their expected, nominal frequencies, but that the *IReferenceClock::GetTime()* method did not. Then, the *QueryPerformanceCounter()* function and the *RDTSC* instruction were implemented in a test program which recorded the clock ticks corresponding to duration generated by an external chronometric computer. Analysis of the counts of the ticks showed that the two techniques have high precision sufficient for submillisecond timing on Winodws PCs.

Key words : サブミリ秒タイム(submillisecond timer), コンピュータ(computer), ウインドウズオペレーティングシステム(Windows operating system)

はじめに

多くの心理学実験において時間を正確に計測すること，および定められた時刻に従って実験を制御することは，その実験の成否を決める重要な要因の一つである．コンピュータを使用して実験制御を行う場合，そのコンピュータ自体が計時や測定に用いられるタイマの役割を果たすことになる．したがって，実験プログラム内で使われる時間関連の関数やメソッドの特性については十分に検討する必要がある．

マイクロソフト社のWindows OS（以下Windows）では時間関連のAPI関数がいくつか提供されている．その中でもミリ秒オーダーでの計時に利用できる周波数分解能を持つ**QueryPerformanceCounter()**関数はコンピュータ制御の実験で広く用いられており，この関数を利用したクラス（MacInnes & Taylor, 2001）も作成されている．

多く心理学実験において想定される計時および制御の時間的オーダーはミリ秒である．しかし，**QueryPerformanceCounter()**関数が参照するカウンターの周波数は1 MHz近くにも達し，より短い時間のオーダーでもこの関数が性能を発揮することができると考えられる．実際，Chambers and Brown (2003) は，適切なプログラムを組めば，この関数を使って多くのコンピュータ上で1 msよりも短い時間を正しく測定できることを示した．

Windows上でサブミリ秒の計時や制御に利用できる方法が別に二つある．一つはPentiumプロセッサの**RDTSC**命令（Intel, 2004）であり，もう一つはDirectXのコンポーネントであるDirectMusicにより提供されるAPIのメソッド（たとえば**IReferenceClock::GetTime()**）である．**RDTSC**命

＊　北海道社会福祉事業団　太陽の園

令では数百MHzという高いカウンター周波数による時間分解能が達成される．一方のDirectMusicのクロック関連メソッドでは，時間分解能はおよそ100 nsである（Microsoft Corporation, 2003）．

本稿では，サブミリ秒オーダーの計時や制御が可能であると考えられるこれら三つ手法の比較・検討を行う．特に，公称値通りの性能を発揮するか，そして精度の高い計時が可能であるのかという点に注目する．

方　法

ハードウェア

テストコンピュータ　各計時手法の性能を調べるためのコンピュータ（以下，テストコンピュータと呼ぶ）としてDell社製ノート型パーソナルコンピュータInspiron 3000が用いられた．CPUはインテル社製Mobile Pentium-MMX（クロック周波数233.86MHz），メモリーは64 MBであり，OSはWindows 2000 Professionalであった．また，シリアルI/Oコントローラ（UART）としてPC16550Dと互換性のある装置が組み込まれていた．

外部計時コンピュータ　外部計時コンピュータとしてNEC社製ノート型パーソナルコンピュータPC-9821 Ne2/340Wが用いられた．CPUはインテル社製86486CPUと互換性のあるAMD社製5x86(クロック周波数133MHz)，メモリは35 MBであった．OSは日本語MS-DOS ver 5.0であった．また，シリアルI/OコントローラとしてμPD8251Aが搭載されていた．

このコンピュータにはタイムスタンパーと呼ばれるカウンタ機能が備わっていた．このカウンタの周波数の文献値は307.2 kHzであり，これは，このカウンタがおよそ3.26μsごとに1回，値を増加させることを意味している（アスキーテクライト，1995；吉田，1994）．

ソフトウェア

テストコンピュータ用プログラム　テストコンピュータ上で使用するプログラム（以下，テストプログラムと呼ぶ）はMicrosoft C/C++ で書かれ，Microsoft Visual Studio 6.0を使ってビルドされた．なお，テストプログラムの振る舞いについては後述する．

QueryPerformanceCounter()関数，**IReferenceClock::GetTime()**メソッド，および**RDTSC**命令によって参照されるクロックの周波数は，公称値でそれぞれ1.193182×10^6，10×10^6，そして233.86×10^6 Hzであった．このうち，**QueryPerformanceCounter()**に関しては，その参照クロックの周波数を得るための関数**QueryPerformanceFrequency()**で調べた値である．また，**IReferenceClock::GetTime()**メソッドのそれは，マイクロソフトの公式プログラマーズリファレンス(Microsoft, 2003)からの値であり，**RDTSC**命令に関してはCPU情報を読み取るシェアウェアによって得られた値である．

テストプログラムの実行時に他のアプリケーションプログラムは実行されていなかった．さらに，バックグラウンドで実行されるプロセスからの影響を最小限に抑えるために，Chambers and Brown（2003）にならい，テストプログラムのプロセスおよびスレッド優先度が最大限に引き上げられた．

外部計時コンピュータ用プログラム　外部計時コンピュータで実行されるプログラム（以下，外部計時プログラムと呼ぶ）はC言語で書かれており，Microsoft C/C++ 7.00を使ってコンパイルとリンクが行われた．この外部計時プログラムは，起動時にタイムスタンパーのカウント数を指定できるようになっていた．このプログラムの振る舞いについても後に述べることにする．

手続き

タイムスタンパーの周波数テスト　外部計時コンピュータに組み込まれたタイムスタンパーの動作周波数を確認するために，このカウンタに同期する形で発生させたパルスの周波数を測定した．具体的には，シリアルポートのピン#7に供される電圧を1000カウントごとに低から高，あるいは高

から低へ変化させ，その周波数をデジタルマルチメータ（Iwatsu Electronic社製VOAC 82）で測定した．

デジタルマルチメータでの測定結果は0.153 kHzとなった．測定された周波数0.153 kHzを，このデジタルマルチメータの測定・表示誤差を考慮してカウンタの更新周期へ換算するとおよそ3.27 ±0.02 μsという値になる．先に示した文献値3.26 μsはこの範囲に含まれており，外部計時コンピュータのタイムスタンパーが公称どおりの性能を有していることが確認された．

テストコンピュータおよびプログラムの事前テスト　各計時手法のパフォーマンスを調べる前に，テストコンピュータとプログラムの事前テストを行った．このテストでは，外部計時コンピュータからの信号に対してテストコンピュータが直ちに応答するプログラムを使用した．このプログラムにはいずれの計時手法も組み込まれていなかった．事前テストは2台のコンピュータをシリアルケーブルで接続した形で実施された．事前テストは4セッションからなり，各セッションは10^4回の試行からなっていた．

各試行は次のような形で進行した．事前テストプログラムは，まず，外部計時プログラムからの開始信号を待ち，それを受け取ると外部計時プログラムへ応答信号を送る．外部計時プログラムは応答を受け取ると，指定されたカウント数分だけタイムスタンパーでカウントを開始する．タイムスタンパーのカウンタがそのカウント数に達すると，外部計時プログラムは事前テストプログラムへ終了信号を送り，そのプログラムからの応答を待つ．一方，事前テストプログラムはこの終了信号を受け取ると外部計時プログラムへ応答信号を送る．応答信号を受け取った外部計時プログラムは，直ちにタイムスタンパーのカウンタ値を読み取り，それを記録する．タイムスタンパーのカウント数は10^2，10^3，10^4，そして10^5であり，一つのセッションではいずれかのカウント数が用いられた．

参照クロックの周波数の確認　各計時手法の参照クロックの動作周波数を確認するために，それぞれの計時手法を使ってシリアルポートの電圧を変化させ，その周波数を前出のデジタルマルチメータで測定した．電圧変化の周期は三つの計時手法の周波数に応じて決められた．**QueryPerformanceCounter()**関数を使った場合の1周期あたりのカウント数は2，**IReferenceClock::GetTime()**メソッドの場合は20，**RDTSC**命令の場合は400とした．

QueryPerformanceCounter()関数の場合，測定周波数は0.5915±0.0015 kHzとなった．1周期のカウント数は2であったので，この関数が参照するクロックの周波数は(1.183±0.003)×10^6 Hzということになる．同様に，**IreferenceClock::GetTime()**メソッドでは(9.09±0.03)×10^6 Hz，**RDTSC**命令では(233.8±0.6)×10^6 Hzがそれぞえの参照クロックの周波数になる．したがって，**QueryPerformanceCounter()**関数と**RDTSC**命令は，参照クロックの周波数は公称値とほぼ同じ値であると言えるが，**IrefrencClock::GetTime()**メソッドのそれは公称値とかけ離れたものとなった．

計時手法のテスト　公称値通りの動作周波数であった**QueryPerformanceCounter()**関数と**RDTSC**命令について，その計時性能を評価するために以下のようなテストを行った．このテストは計時手法ごとに4セッションからなり，各セッションは10^4回の試行で構成されていた．各セッションでは，あらかじめ決められた一定数を外部計時プログラムがカウントする条件で，毎試行，以下のような測定が行われた．なお，それぞれのセッションには1×10^3，2×10^3，3×10^3，4×10^3，あるいは5×10^3のいずれかのカウント数が用いられた．

テストプログラムは，外部計時プログラムからの開始信号を受け取って応答信号を送り，直ちに計時手法ごとの参照クロック値を記録する．終了信号を受け取った場合も同様に，応答信号を送ってからすぐさま参照クロック値を記録する．そして，終了時のクロック値から開始時のクロック値

を引いた値を各計時手法ごとに計算し記録する．その他の振る舞いは前述の事前テストプログラムと同じである．

結果と考察

計時における偏差

テストプログラムに組み込まれた計時ルーチンによって各計時手法に応じたカウンタ値が記録された．ここでは，測定値に関して簡単なモデルを設定し，実測値と期待される値との差（偏差）を分析する．

理論値　タイムスタンパーの指定カウント値をνとする．その値は1×10^3，2×10^3，3×10^3，4×10^3，あるいは5×10^3のいずれかである．また，ある計時手法のカウンタの周波数をφHz，タイムスタンパーのそれをψHzとすると，その計時手法を使ったときに期待されるカウンタ値$\hat{t}$は

$$\hat{t}=\nu\varphi\psi^{-1} \qquad (1)$$

である．以降，この$\hat{t}$を理論値と呼ぶことにする．

実測値　しかしながら，実際に記録されるカウンタ値は式(1)のようにはならない．なぜなら，測定には必ずランダムな変動要因が影響を与えるからである．さらに，二つの周波数φとψの値に，実際のプログラムにインプリメントされることによる影響は考慮されていない．今回の計測では，タイムスタンパーのカウンタおよび計時手法側のカウンタのいずれに対しても，それらの値をプログラムで読み出すという処理が行われている．これらの処理は極めて高速に行われるはずであるが，それでもごくわずかな遅延（おそらく1%にも満たないだろう）は避けることはできない．

これらを考慮した形で，計時手法側で実際に記録されるカウンタ値（実測値）をモデル化する．まず，計時手法側のカウンタの周波数で遅延を考慮した値を$\alpha\varphi$Hz（αは定数）で表す．定数αは1未満の（しかしながら1に大変近い）値をとるものとする．これにより，測定時における周波数は理論値あるいは公称値よりも若干小さな値となり，カウンタ動作の遅延を表現することができる．同様に，タイムスタンパーのカウンタ周波数で遅延を考慮した値を$\beta\psi$Hz（βは定数）とする．このβにも上記のαと同様の仮定を設ける．そして，これらを使って計時手法側のカウンタの実測値tを表すと，

$$\underset{イタ}{t}=\nu\,\alpha\varphi\beta^{-1}\psi^{-1}+\underset{イタ}{e} \qquad (2)$$

となる．ここでeはランダム因子であり，その期待値はゼロであるとする．なお，各カウンタの動作には遅延の影響を考慮したが，それでもtが$\hat{t}$よりも大きくなるとは限らない．それらの関係はαとβの値によって決まる．

偏差量のモデル化

データにもとづいた見積もり　計時手法側で見られる理論値からのずれの大きさ（偏差量）を見積もるために，理論値$\hat{t}$に対する実測値tの比を検討する．式(1)と(2)の比をρとすると，それは

$$\begin{aligned}\rho&=\frac{\underset{イタ}{t}}{\underset{イタ}{\hat{t}}}\\&=\frac{\nu\,\rho\varphi\beta^{-1}\psi^{-1}+e}{\nu\varphi\psi^{-1}}\\&=\alpha\,\beta^{-1}+e\end{aligned} \qquad (3)$$

となる．ただし，$e/(\nu\varphi\psi^{-1})$を改めてeとした．

ρの値を測定データから見積もってみると，タイムスタンパー側の指定カウント数によらず平均で，**QueryPerformanceCounter()**関数の場合およそ0.999，**RDTSC**命令の場合およそ1.0002となった．仮定より，比ρのうち式(3)のeで表される部分の平均は0であるので，これらの値はそのまま$\alpha\beta^{-1}$の値であると考えて差し支えないであろう．そして，どちらの計時手法でも偏差量は1%に満たないことがわかる．

測定周波数からの見積もり　この$\alpha\beta^{-1}$の値は別の方法でも見積もることもできる．それには，

各計時手法あるいはタイムスタンパーの公称周波数と，デジタルマルチメータを使って得られたそれらの実測周波数との比を利用する．いままでの議論に即して述べると，前者はφあるいはψに，後者は$\alpha\varphi$あるいは$\beta\psi$に相当する．これらの比からαとβをそれぞれ計算し，その値を使って$\alpha\beta^{-1}$を求めるのである．

まず，計時手法側の遅延を表す係数αを求める．**QueryPerformanceCounter()**関数および**RDTSC**命令の公称カウンタ周波数は，それぞれ1.193182×10^6 Hzおよび233.86×10^6 Hzであった．また，実測周波数はそれぞれ$(1.183\pm0.003)\times10^6$ Hzおよび$(233.8\pm0.6)\times10^6$ Hzであった．これらの値から公称周波数に対する実測周波数の比を誤差を考慮して計算すると，**QueryPerformanceCounter()**関数では0.991 ± 0.003，**RDTSC**命令では0.9997 ± 0.0025となる．これらがαの値に相当するものであり，前者をα_p，後者をα_rとする．

次にタイムスタンパー側の遅延係数βを求めるのであるが，ここでは周波数の逆数，つまりカウンタの更新周期の比を計算することにする．タイムスタンパーのカウンタの更新周期は，公称値で3.26 μsつまり3.26×10^{-6} sであった．また，実測周波数から換算した値は$(3.27\pm0.02)\times10^{-6}$ sであった．後者に対する前者の比がβに相当し，その値は0.997 ± 0.006となる．この値をβ_tで表す．

これらの値を使って，改めて式(3)の$\alpha\beta^{-1}$を誤差の部分を除いて計算すると，

$$\alpha\beta^{-1}=\begin{cases}\alpha_p\beta_t^{-1}=\dfrac{0.991}{0.997}\fallingdotseq0.993\ (\text{QueryPerformanceCounter()関数})\\ \alpha_r\beta_t^{-1}=\dfrac{0.9997}{0.997}\fallingdotseq1.0002\ (\text{RDTSC命令}),\end{cases}$$

となり，いずれも先にデータから算出したものと近い値となった．

まとめ

QueryPerformanceCounter()関数および**RDTSC**命令による計時では，理論値との誤差は1%にも満たない大きさであった．また，これら二つの手法は公称どおりの動作をしていた．したがって，Windows PCで計測や制御を行う場合にはいずれかの手法を利用するのが好ましい．

一方，**IrefrencClock::GetTime()**メソッドでは実測周波数と公称周波数が大きく異なっていた．公称周波数での動作が保証されないのであれば，使用するたびに外部の装置を使って動作周波数を確かめることが必要になる．したがって，計時・制御にこの手法を用いるのはできるだけ避けたほうがよいであろう．

ところで，測定値にはランダムな変動が含まれているが，理想的にはその変動は正規分布に従っていることが望ましい．詳細は省略するが，タイムスタンパー側の残差はこの性質を有しているようである．一方，各計時手法に起因する残差の分布については満足のいくような解析結果には至っていない．今後はこの点を中心に分析を進めてゆきたい．

引用文献

アスキーテクライト（編）(1995). PC-9800シリーズ テクニカルデータブック CD-ROM版 HARDWARE編 第2版 アスキー出版局

Chambers, C. D., & Brown, M. (2003) Timing accuracy under Microsoft Windows revealed through external chronometry. Behavior Research Methods, Instruments, & Computers, 35, 96-108.

Intel Corporation (2004) IA-32 Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide.

MacInnes, W.J., & Taylor, T.L. (2001) Millisecond timing on PCs and Macs. Behavior Research Methods, Instruments, & Computers, 33, 174-178.

Microsoft Corporation (2003) Microsoft DirectX 9.0 SDK Update (Summer 2003) Programmer's Reference. <http://www.asia.microsof

t.com/japan/msdn/directx/downloads.asp> (2004年5月19日)
吉田　功 (1994) トランジスタ技術SPECIAL No.45 特集 PC98シリーズのハードとソフト－386 & 486マシンを使いこなす－CQ出版