

HACSTIP Ver. 2 の開発とそれを用いた時刻補正の精度の検証

2023. 6. 10（時の記念日） 宮下和久

概 要

GPS を活用した PC 時刻管理ソフトウェア HACSTIP*³を、高精度の関数を実装するなどして改良した。また、それを用いて、これまで掩蔽観測に用いられてきた機器やソフトウェア、解析手法について検証を行ったので報告する。

なお、長文となるので、実際の観測やソフトウェアの設定などの場面で必要となることについて、下記にまとめて示す。図番号等で検索するなどして活用いただきたい。

索 引

新たに明らかになったこと、および 考察のなされたこと。

1. 従来の GPS 利用の時刻補正ソフトウェアで「精度よく補正できている」と思われていたが… GetSystemTime 関数の誤差と遅延の値が大きく、精度は期待されたほど高くなかった。

- Garmin18xlvc および GHS-OSD のような DSR からの 1PPS 信号を利用できる受信装置は
 - ・平均 8msec 程度の遅延と、最大+/-8msec 程度の値のばらつきがある。 [図 3]
 - ・用いる USB 変換器により、更に 2msec の遅延と 2msec のばらつきが加わる。 [図 11]
- VK172 や VFAN UG353 のような、USB に直接接続するような市販の受信機は 50msec から 100msec ほどの遅延があり、遅延量も時間とともに変化する [図 9]
 - ⇒ 観測に HACSTIP の常時補正 ON で用いることは誤差が大きく不適切。

2. HACSTIP Ver.2 による時刻補正

- GetSystemTimeAsFileTime 関数と SetSystemTimeAdjustment 関数 [図 4]
- DSR 通信線を介した 1PPS 信号の検出精度 [図 10]
- 時刻補正の精度の考察（推定平均遅延量 0.14msec, 1σ 0.06~0.10msec） [図 18]
 - ⇒ 誤差が 1 ミリ秒よりはるかに小さいことから、DSR タイプでは常時補正が可能。
- 補正精度は高いが、これにキャプチャソフトの処理遅延が加わるので、いずれにせよ LED の録画は必要。なお、明瞭に LED を録画できれば、現象前または後の一方でよい。 [図 17]

3. 従来の諸ソフトウェアの補正精度（1 項に記した内容に加えて）

- HACSTIP Ver.1
 - ・ VK172 などについて、常時時刻補正が可能であり、誤差要因となっていた。
 - ⇒ Ver.2 を利用して 観測前 1 回の時刻補正(One time correction)とすること。
- Satk
 - ・ DSR との差が 30msec を超えたときに時刻補正がなされる。常時補正ではない。 [図 20]
- GPS-Clock
 - ・ 動作シーケンスについて、現在検証中。
- さくら時計
 - ・ およそ+/-5msec の精度で補正されるが、まれに 20msec を超える誤差もある。 [図 21]
 - ⇒ 常駐や、起動したらオンライン、にはしないこと。
観測前に 1 回の時刻補正とすること。

はじめに

HACSTIP Ver.1.x.x.x *³は、PC システムタイムの取得と設定にそれぞれ、Windows が API として提供する次の関数を用いていた。

時刻取得	GetSystemTime	関数* ¹¹	(時間分解能 1msec, 取得精度 15msec のばらつきあり)
時刻設定	SetLocalTime	関数	(時間分解能 1msec, 設定精度 不明)

ここで掩蔽・星食観測の将来への展望を考えると、観測結果として要求される現象時刻の精度は ± 1 ミリ秒あるいは場合によってはそれを超える精度の値が要求されるようになる可能性がある。しかし PC 時刻の保持が上記のようであるならば、それに対応することは困難な状況となってきた。そこで使用する関数を以下のものに変更して将来にわたって活用可能な補正ソフトウェアとなるように改良を行った。

時刻取得	GetSystemTimeAsFileTime	関数* ¹⁰	(時間分解能 100nsec, 取得精度 およそ $10\mu\text{sec}$)
https://learn.microsoft.com/ja-jp/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsystemtimeasfiletime			
時刻設定	SetSystemTimeAdjustment	関数	(時間分解能、設定精度 とともに $1\mu\text{sec}$ とされている)
https://learn.microsoft.com/ja-jp/windows/win32/api/sysinfoapi/nf-sysinfoapi-setsystemtimeadjustment			

その結果、従来(GetSystemTime)が 1 ミリ秒単位での時刻取得であったのに対し、新バージョンでは 0. 1msec 程度またはそれより小さな誤差での時刻取得が可能となり、図 1 のように PC システムタイムと GPS から得た UTC との差も小数点以下 2 桁の数値で表示することも意味のあるものとなった。

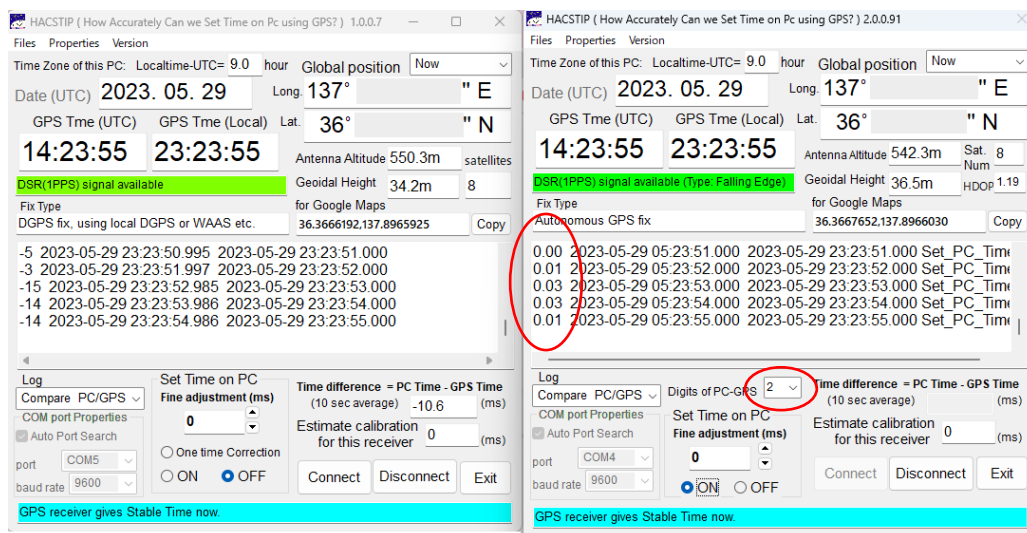


図 1 HACSTIP の旧バージョン (左) と新バージョン (右)

本稿では、精度の高い時刻取得関数を用いた新バージョンを用いて PC システムタイムの測定を行うとともに、様々な GPS 受信機による時刻補正の様子や精度についても測定して、精度がどの程度向上・改善されたかについて測定と検証を行った。その結果、実際の観測に活用できる知見が得られたので、HACSTIP の新バージョンの機能の紹介も合わせて報告する。

1. GPS 利用の時刻補正ソフトウェアの動作のしくみ

(1) 一般的な GPS 受信機を利用した時刻補正のしくみ

一般的な GPS の位置測定や時刻補正への利用は、RS232C を介して接続された受信機が毎秒出力する NMEA センテンスを読み込んで行われる。NMEA には様々な情報を伝えるために多くのセンテンスが用意されているが、位置と時刻の基本的な情報を知るためには、\$RMC と\$GGA の 2 行があればよい。RMC 行からは日付と時刻を、GGA 行からは経緯度と標高関係および使用衛星数を得ることができる。

表 1 NMEA センテンス (例)

\$GNRMC,082928.00,A,36**.*,N,137**.*,E,0.078,,050623,,D*62
\$GNGGA,082928.00,36**.*,N,137**.*,E,2,12,0.74,556.4,M,36.5,M,,0000*47
\$GNGLL,36**.*,N,137**.*,E,082928.00,A,D*76

文の出力順序については、正秒の直後に RMC 行が出力され、それに続いて様々な文が出力される。VK172 のような NMEA センテンスのみが得られるような受信機を使用した場合、時刻補正ソフトウェアはこの RMC 行を受け取った瞬間を正秒として扱い、時刻補正を行っている。従来は「RMC 行は正秒から一定の遅延をもって出力されている」とされていたことから、従来の時刻補正ソフトウェアではその値を時刻値に加えて PC に設定（またはそれに相当する進み方の補正）をすることで時刻の補正が行われていた。しかし、この遅延については、一定であるのか、また機種によって異なるのか、などの検証はなされてきていない。

(2) 受信機の 1PPS 信号を RS232C の DSR ピンを介して伝える受信装置

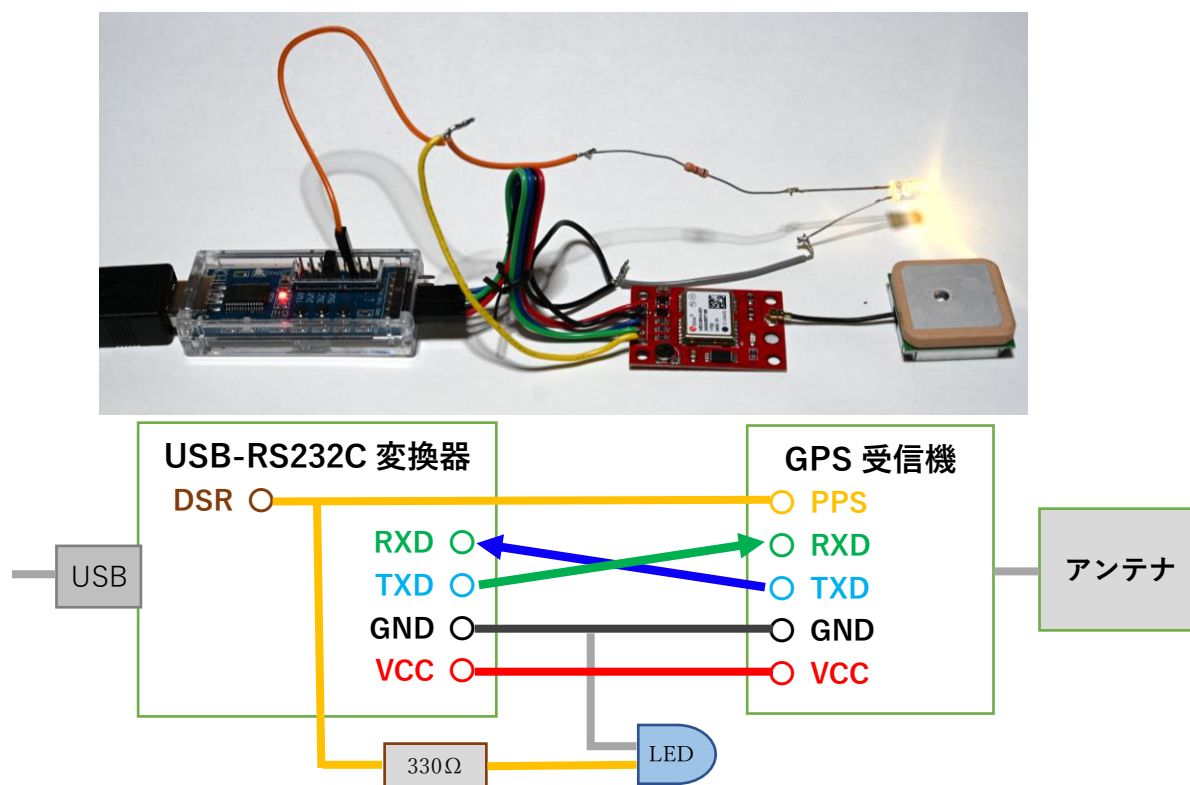


図 2 RS232C-USB 変換器 (左) と受信機 (右) の配線の例

撮影したビデオファイルの時刻補正のために 1PPS を LED を接続するとともに DSR にも接続。受信機と変換器の TXD(センテンス等出力)と RXD (センテンス等入力) はクロス配線とすること。

2000 年頃、RS232C の DSR 制御線に GPS の 1PPS 信号を入力して PC に伝えることで、より高い精度での時刻補正を行うことができる装置が考案され、そのための時刻補正ソフトウェアとして Satk（さとくん）が故 瀬戸口貴司氏によって開発された*¹。図 2 に装置の基本的な配線を示す。ソフトウェア側から DSR 通信線の信号の変化（立ち上がりまたは立下り。変換器の機種によって異なる。）を検出し、その瞬間を正秒として扱って補正が行われる。シリアルドライバはシリアルポートのプロパティ（値）として受信した DSR 信号をセットする。制御用の通信線による信号であることから、ソフトウェア側からの正秒の検出も高速で精度よく行うことができる。このような装置を用いたときは、HACSTIP も同じ仕組みで時刻補正を行う。

この仕組みを利用した場合の時刻精度については第 3 項で述べる。

2. PC システムタイムの変化と取得関数における時刻精度

(1) GetSystemTime 関数（旧）*¹¹ および GetSystemTimeAsFileTime 関数（新）*¹⁰ で見る PC の時刻

PC システムタイムは、ほぼ一定程度の速さでの進み遅れを持っている。いったん時計を UTC に合わせても、数時間とか 1 日とかが経過すれば、UTC からの外れが起こってしまうのがふつうである。

その様子は、これまでは HACSTIP により GPS に同期して UTC を基準として PC システムタイムの進み遅れを調べることににより明らかにされてきた。しかしそれは、GetSystemTime 関数を利用したものであり、ある程度の時刻のばらつきがあったために、正確にとらえることはできなかった。そこで、HACSTIP の機能を使った試験用のバージョンを作成し高精度の GetSystemTimeAsFileTime によって測定するとともに、GetSystemTime の精度についても同時に調べることにした。結果を図 3 に示す。

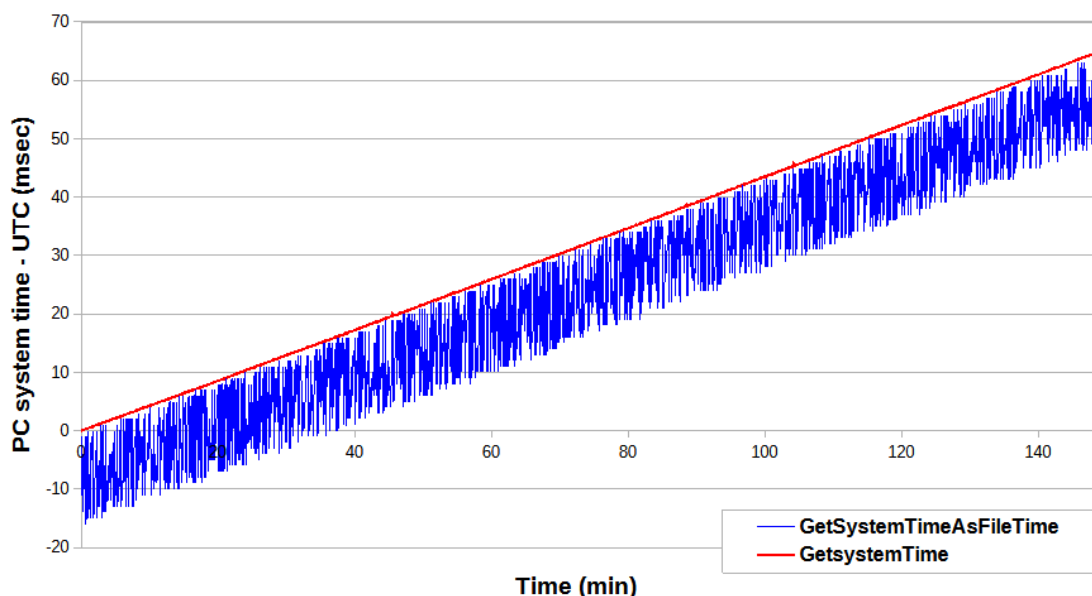


図 3 GetSystemTime 関数（青）, GetSystemTimeAsFileTime 関数（赤）から見た PC システムタイム

これによると、実験に用いた PC のシステムタイムはほぼ一定の割合で「進む」傾向となっており、そのようすは GetSystemTimeAsFileTime 関数によりなめらかな直線としてとらえられている。一方、GetSystemTime 関数では、時刻取得値に 15msec 程度の変化が見られ、また、GetSystemTimeAsFileTime 関数による読み取り値よりも、平均して 8msec ほどの「遅れ」が見られる。更には、GetSystemTime の取得値は、GetSystemTimeAsFileTime の値よりも「進んでいること」はなく、必ず「遅れている」。

この原因について考えるために、HACSTIP が受信機に接続した状態におけるそれぞれの関数の処理に要する時間を測定した。結果を表 1 に示す。なお、双方の関数は実行速度に大きな差はない。

表 2 GetSystemTime 関数 および GetSystemTimeAsFileTime 関数の処理に要する時間

GetSystemTimeAsFileTime 関数* ¹⁰	
1000 回の実行に要した時間: 855, 839, 632, 826, 832, 832, 821, 831, 831, 823	(x 100 nsec)
平均 81000nsec / 1000 回	⇒ 1 回あたり、81nsec
GetSystemTime 関数* ¹¹	
1000 回の実行に要した時間: 1111, 1195, 1043, 1040, 1040, 1231, 1044, 1166, 1040, 1046	(x 100 nsec)
平均 109560nsec / 1000 回	⇒ 1 回あたり、110nsec

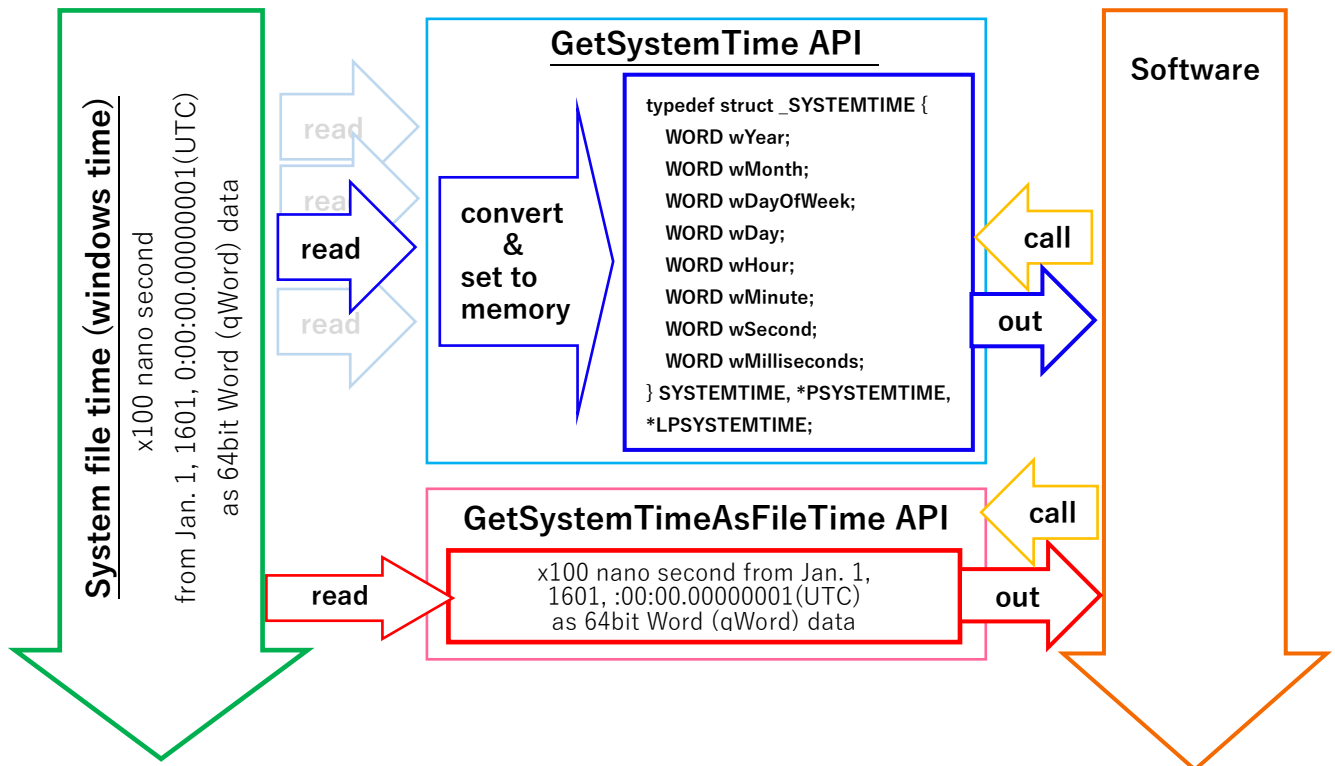


図 4 実験結果から推定した各関数 (API) の動作

測定結果から考えると、双方の関数の動作のしくみは図 4 に示すようであろうと考えられる。

Windows がシステムタイムとして保持・管理しているのは、System file time と呼ばれる 1601 年 1 月 1 日 0:00:00.00000000 を起点とした 100nsec を単位とした 64 ビットの数値である。GetSystemTimeAsFileTime 関数はその値をそのまま取得し、戻り値としてソフトウェアに提供する。この作業はコールされたときに瞬時に行われることから、システムタイムを高精度で読みだすことが可能となる。また、システムタイムを年月日や時分秒に変換するのは、ソフトウェアが行うことになる。

一方、Windows API の GetSystemTime 関数はこの 64 ビットの値を、年月日、時分秒、ミリ秒等の値に変換し、それらの値の集まりである Systemtime 構造体に格納する*¹¹、という作業を随時（ある程度曖昧な周期性をもって）行っており、時計表示などの用途に提供している。ソフトウェアは GetSystemTime 関数を通じて受け取った構造体の値を利用すれば、うるう年などの複雑な変換を行うことなく、そのまま表示等を行うことができる。ただ、上記のように API がいつ読んだ時刻なのかは明確ではなく、そのため「システムファイルタイムから最大 15 ミリ秒遅れた、15 ミリ秒の幅をもつ時刻」が得られることになってしまう。

以上が、呼び出しから戻り値を得るための所要時間が両関数でほとんど差がないにも関わらず、GetSystemTime から得られる時刻値に遅延やばらつきがあることの原因として考えられる仕組みである。

また、実験の結果から「GetSystemTime 関数を用いた、HACSTIP Ver.1.x.x.x を含む従来の時刻管理ソフトウェアは、最も正確な状態でも UTC に対して平均 8msec ほど遅れた時刻を提供していた。」ということが明らかになった。

(2) GetSystemTime 値のばらつきが小さくなる場合もある

これまで HACSTIP で PC システムタイムの時刻値を連続的に見ていくと、時々変動が小さくなる期間があった。また、観測用ビデオキャプチャソフトウェアである SharpCap を起動し、カメラを接続してプレビューや録画が始まると、その期間だけ変動が小さく収まることも、観測者の間で指摘されていたことであった。図 5 は、そのようすを見やすく表したものである。具体的には、HACSTIP を二つ起動し、一方は管理者モードにて GetSystemTimeAsFileTime 関数および SetSystemTimeAdjustment 関数を用いた精密な時刻補正を行い、もう一方は GetSystemTime にて、時刻取得を行った

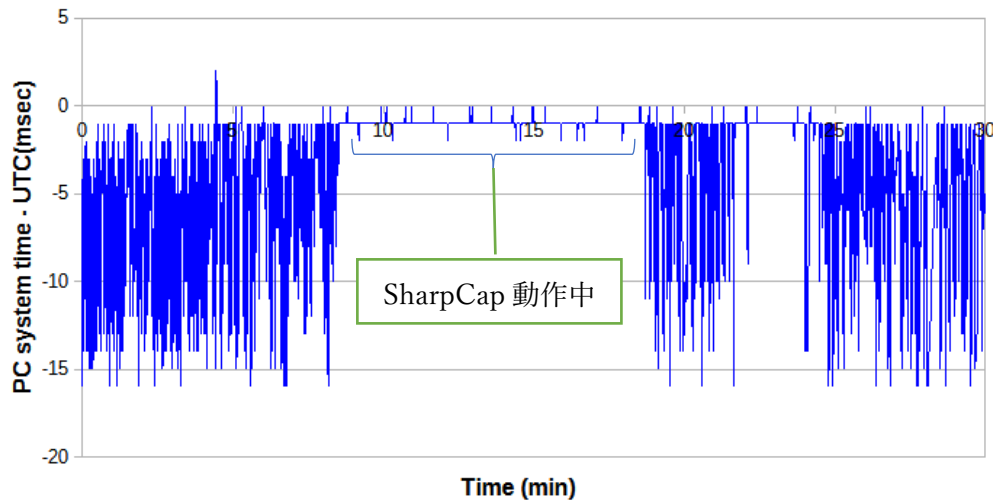


図 5 GPS 受信機からの 1PPS 信号取得時に PC システム時刻を読み込んだ値と 1PPS (UTC) 信号との差 (HACSTIP 改良前の、GetSystemTime および SetLocalTime 関数を使用していた時)

キャプチャソフトウェア SharpCap は、各フレームに記録される時刻値が小数点以下 7 桁で表現されていることから、GetSystemAsFileTime 関数を用いて時刻取得し、それを西暦 1 年 1 月 1 日 0:00:00.00000001 を起点とした値に変換して用いていると考えられる。GetSystemTimeAsFileTime の実行に伴って GetSystemTime API の処理の優先順位が高くなり、そのために誤差の少ない値を読み取ることができるのではないかと考えられる。同様に、PC 内で動作するプロセスの変化により、GetSystemTime 関数の優先度が高くなれば、精度の良い時刻読み取りが可能となるのであろう。ただしこれについては、PC の動作環境の変化の詳細を見ることはできないことから、明らかにすることは困難である。

3. DSR 信号検出の精度

GetSystemTimeAsFileTime 関数により、高精度にシステムタイムの時刻値を得ることができることは前項で述べた。したがって、時刻補正における精度は、DSR 信号検出の時刻精度に依存するところが大きいといえることができる。それでは、DSR 経由の 1PPS 信号は PC 上で動作するソフトウェアの側から、どの程度の時刻精度で検出することができるのだろうか。従来、精度の高いシステムタイム読み込み関数が利用できなかったことや、検証用のソフトウェアの開発もされていなかったことから、それを確かめることは困難であった。そこで、新たに、HACSTIP の DSR 信号検出ルーチンを活用して検証を試みることにした。

(1) SerialPort.DsrHolding プロパティ*12 と シリアルドライバ

RS232C の DSR 信号は、PC においては COM ポートを介して受信される。この通信を受け持つのがシリアルドライバであり、RS232C-USB 変換器が PC に接続されるとドライバが (インストールされていないときはインストールされ) 動作を始める。DSR 信号の受信の仕組みは、図 6 のようであると考えられる。

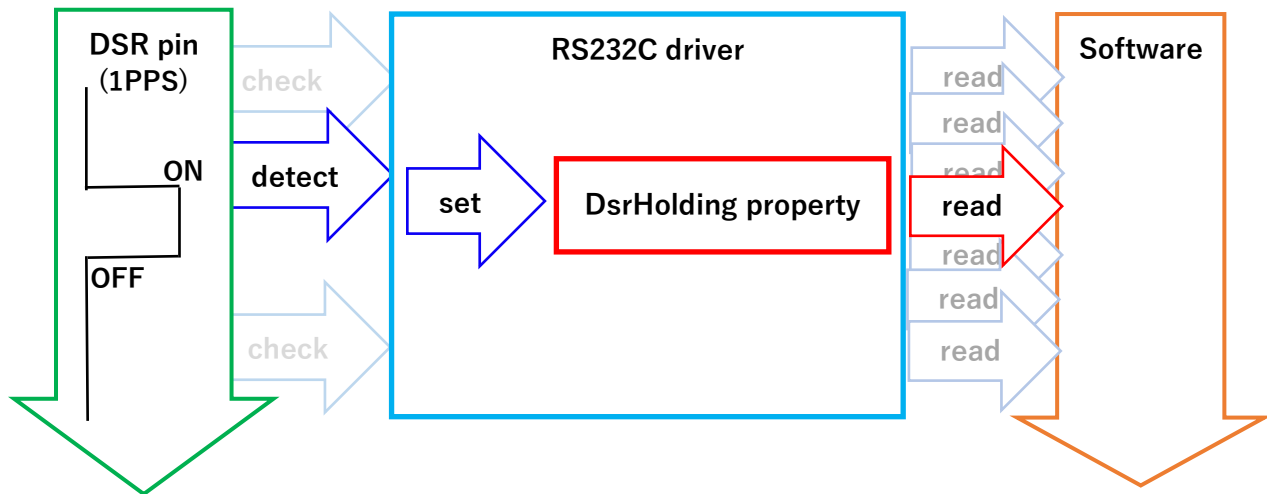


図6 DSR 信号（状態）の検出のしくみ（推測）

(2) DsrHolding property の読み込みの動作の様子

図6の右側に当たる、ソフトウェアからのDsrHolding property の読み込みの動作の様子を調べてみた。

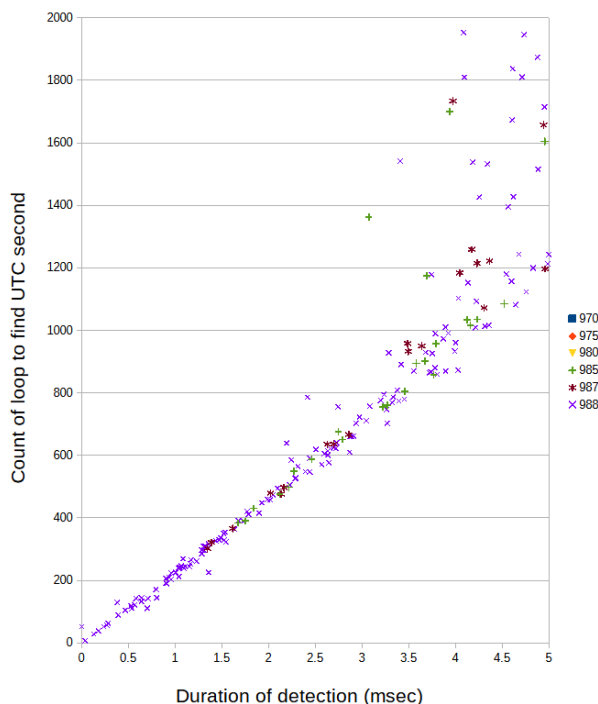


図7 1PPS 検出期間と検出ルーチンの繰り返し回数

HACSTIP は、正秒(1PPS)の数十ミリ秒前から SerialPort.DsrHolding プロパティ*12 を調べる処理を繰り返して行い、正秒を示すプロパティ値を検出次第、時刻補正の処理を行う。このとき、他の処理にかかる時間のばらつきから、正秒を探す繰り返し（ループ）処理の開始タイミングが前後にずれ、その結果探す処理の継続時間が様々な異なることになる。そのとき、1回の SerialPort.DsrHolding プロパティを調べる処理にかかる時間が一定であるかどうかをまず調べる。

図7は、HACSTIP の正秒検出ルーチンが検出期間の長さ（msec）に対して何回「回った」かを表したものである。これによると、検出期間が3.5msec よりも短い期間であったときには、検出時間と検出回数がほぼ比例している。これは、DsrHolding プロパティを読み出す関数 (getdsr) の動作時間が一定（およそ5 μ sec）であることを示している。

また、検出期間が3.5msec を超えると回数が多くなるが、これは、プロパティにセットされた値を読む処理であるため、メモリの先読みキャッシュがはたらか、処理が高速化されたためと考えられる*8。

次に、DSR の変化を検出した直後の時刻を正秒ごとに測り、DSR の1PPS 信号の検出から次の正秒での検出までの時間（以下、DSR 検出の間隔と略称）を求めることを繰り返して行うプログラムを作成した。

結果を図8に示す。これによれば、DSR 検出の間隔は最大0.1msec の幅をもって変化する。1 σ は0.06msec ほどの変化である。

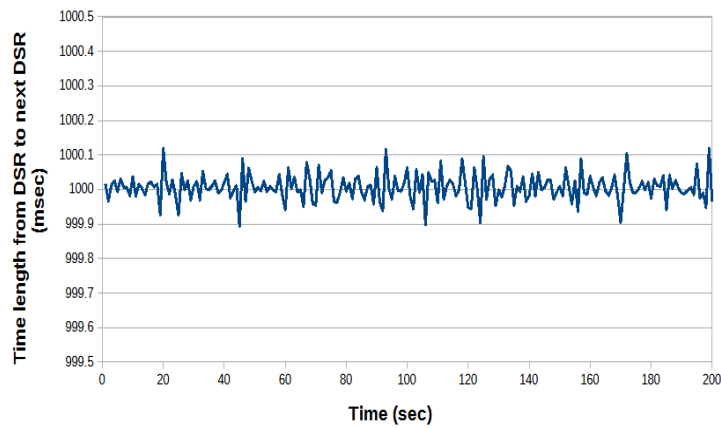


図 8 1PPS を示す DSR 値の間隔の変化

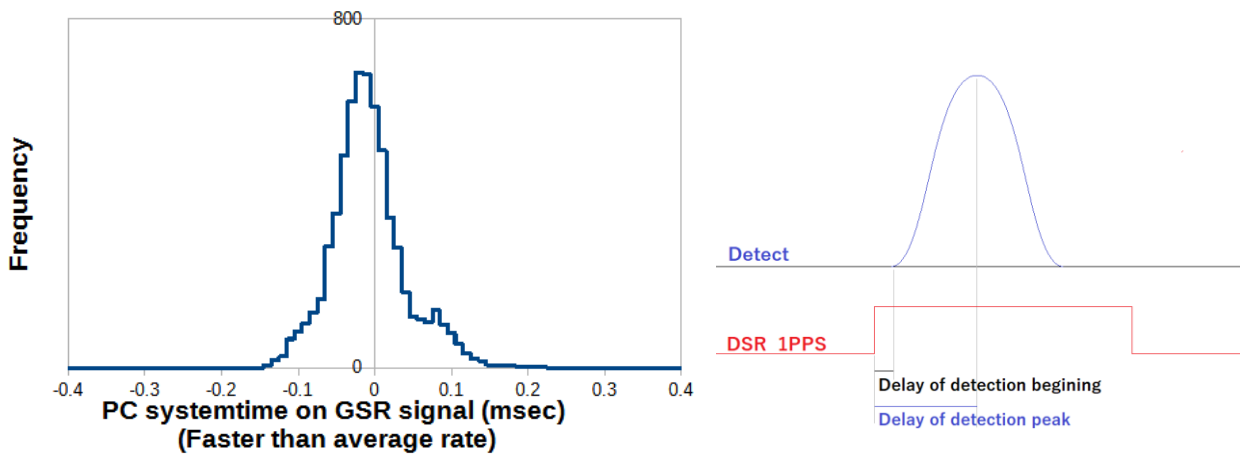


図 9 DSR 検出時刻のばらつき (System time の平均的進みとの差)

左： 測定結果 右： 概念図

また、時刻誤差について直接的に把握するために、時刻の平均的な進み方に対する各正秒の時刻値のばらつきについても調べた。進み遅れのある PC システムタイムについて、DSR の信号を検知した瞬間を正確な UTC 正秒であると仮定してそのときのシステムタイムの時刻値を得、そこから仮定した UTC 正秒の値を求めた。多数の正秒で得たシステムタイムの時刻値とこの平均の時刻の差について、分布の様子を表したのが図 9 である。なお、図 9 左図の X 軸の 0 は、検出時刻のすすみの平均値を示すものであり、UTC を示すものではない。値の分布はほぼ正規分布に近く、そこにデータ取得が遅れた場合と思われる+側の小さな山が重なっている。

この結果より DSR 信号の変化の瞬間の時刻を得る処理のようすについて以下のように推定した。

DSR 信号の変化に対するシリアルドライバの応答（割り込み発生と DsrHolding のセット）については、DSR 信号は ON/OFF のみの単純なデータであるから、割り込みの検出については多くの時間を要するものではないと考えられる。しかし Windows は RTOS（リアルタイムオペレーティングシステム）ではないので、システムも含め、ハードウェア、ソフトウェアからの多数の割り込みを処理しているため、より優先される割り込みが実行中であれば DsrHolding のセットも遅れることになる。

八木(2003)*6 は、放射線計測機の信号処理に関して、Linux, RTLinux, Windows など PC に負荷を掛けない場合について割り込み応答速度を測定し比較し、その結果から、応答速度は、Windows の場合、4~17 μsec のように幅が広いことを指摘している。一般の用途に使用される Windows は、複数の周辺機器が接続されていることから、ハードウェア割り込みだけでも多数の割り込みが発生している。それらは全体から俯瞰すればタイミング的にはランダムに起きていると見ることができると、ソフトウェアが検出しようとする割り込みに優先する割り込みも多数あるはずだから、DsrHolding への値セットは 1PPS ごとにある幅を持

った遅延が発生することになる。それが図 9 の幅のある分布となって現れていると考えることができる。

以上のような仕組みであるならば、DSR の信号変化に対する応答時間は、ある一定の幅を持ち、統計的な分布すなわち正規分布に近い分布を示すことになる。また、その中には「他に優先する割り込みがなく、優先的に処理がなされる場合」が少数ながら存在することになる。その場合には、DSR 信号が変化した直後に間を置かずに割り込みとデータセットがなさるであろう。図 9 左図において、0.13msec が下限でありそれよりマイナスの値（早い時刻）が見られないことも、このことを裏付けていると考えられる。

また、上述のように DSR 信号の ON/OFF の値をセットするにはほとんど時間がかからないであろうと考えられるから、図 9 の示す-0.13msec が DSR 信号の変化の瞬間であると見てよいであろう。換言すれば、図 9 右の Delay of detection beginning がほぼゼロに近いであろう、ということである。

分布のピークは+0.01msec 付近であるから、ピークの位置は DSR からの UTC 信号に対して 0.14msec 遅れた時刻であると考えられる。なお、このことについては、今後の更なる検証が必要である。

また、ばらつきについては、1 σ として 0.06msec から 0.10msec が測定から得られており、これが時刻取得の安定性を示す値であるとすることができる。

(3) 精度が変換器やドライバに依存することを示す事例

安価に GPS 受信装置を自作する方法の草分け的な存在となったのが、2019 年に渡辺裕之氏により開発された装置である*2。受信機 Garmin18xlvc を DSub9 ピンコネクタを介して RS232C-USB 変換ケーブルに接続する方法である。Garmin18xlvc は図 10 に示すように、精度よく 1PPS で DSR 制御線から PC に届けることのできる機種である。

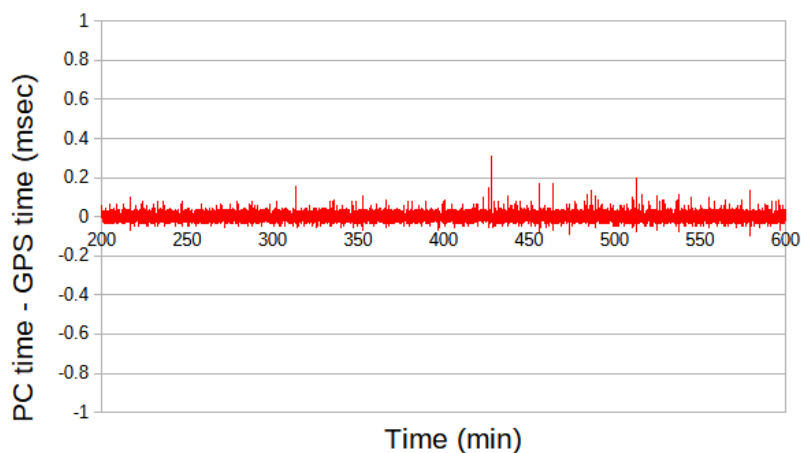


図 10 Garmin18xlvc による時刻補正結果（FTDI 製 RS232C-USB 変換器 FT232RL 使用時）

平均 +0.001msec 標準偏差 0.027msec と、高精度の補正がなされている

なお、HACSTIP の時刻補正機能を用いた実験の結果であるが、精度の検証については後に詳述するので、そちらを参照いただきたい。

ところが、同じ Garmin18xlvc であっても、変換ケーブルによっては DSR の 1PPS 信号を検出したときの時刻値が数ミリ秒の範囲でばらついてしまう、という現象が確認された。これまでの GetSystemTime 関数を利用したソフトウェアでは気づくことができなかったが、精度の高い GetSystemTimeAsFileTime 関数を用いたことで明らかにすることができたのである。図 11 にその様子を示す。

実験の方法は、ひとつの HACSTIP を受信機 GT902PMGG に接続して時刻補正を行い、PC システムタイムを精度よく管理した。その状態で、もう一つの HACSTIP を起動し、SABRENT 銘変換ケーブル（基板は Prolific PL2303GS）を介して Garmin18xlvc に接続した。これにより、高い精度で補正された PC システムタイムで、この変換器を使用した時の時刻のばらつきのようすを見ることができる。

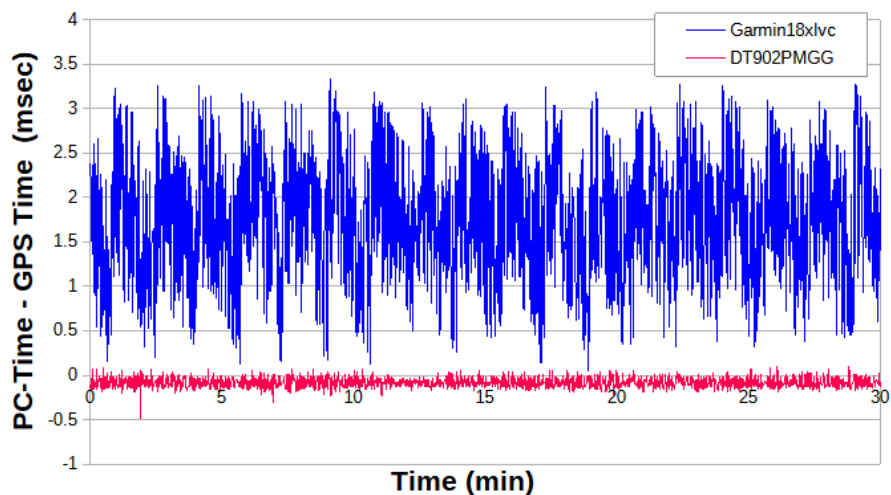


図 11 DSR 信号の検知を要求したときの戻り値が返るまでの時間

青色： Garmin18xlvc を SABRENT 銘変換ケーブル（基板は Prolific PL2303GS）に接続した場合

赤色： 受信機 GT902PMGG を変換器 DSD TECH 製 SH-U09C5 に接続して PC 時刻を補正

図では、得られた時刻が時計でいうところの「遅れ」となっているときに+の値として示している。

この遅延とばらつきの原因について、HACSTIP のメンテナンス用情報機能の結果を元に考察する。

表 3 HACSTIP の動作の記録（一部）

変換器銘柄・機種名 ドライバ 受信機機種名 Sleep 開始 正秒の xxx.x ミリ秒後 Sleep 終了 正秒の xxx.x ミリ秒後 DSR=OFF 検出 i= x 回 DSR=ON 検出 i= x 回	Prolific PL2303GS Prorific Garmin18xlvc Before sleep789.6 Wake up 972.0 Pre DSR=OFF i=1 Post DSR=ON i=6	FT232RL モジュール FTDI GHS-OSD Garmin15x Before sleep884.4 Wake up 968.6 Pre DSR=OFF i=1 PostDSR=ON i=11265
--	---	---

表 3 は、ある時点(秒)における「正秒の検出」の状況を示したものである。これによると GHS-OSD(FTDI 製変換器・ドライバ)や GT902PMGG (DSD TECH 製、FTDI ドライバ)などの受信機の場合、Sleep 終了後の 30msec 間に DSR の状況を検出するサブルーチンが 10000 回ほど繰り返しながら正秒の状況となるのを待っている。ルーチン 1 回の所要時間は約 $3\mu\text{sec}$ である。一方、Prolific 製変換器に接続した Garmin18xlvc では、30msec の間に DSR 検出のサブルーチンが計 7 回で、1 回の所要時間は約 4msec である。Prolific 社の IC チップおよびドライバは、TXD および RXD を通じて行われるデータ通信についても他社と異なる性質があることも報告されている*4 ことから、DSR についても取得の状況が独特のものとなっていると考えられる。いずれにせよ、このように時間分解能が低くなってしまう場合には精密な時刻補正には適さない。これを見分けるには、図 12 のように、スタートメニューからデバイスドライバを選択し、COM ポートを選んでプロパティのドライバタブで確認することができる。



図 12 時刻補正に適した変換器であるかの見分け方

(4) DSR 信号の時刻取得の遅延とばらつきの大きい変換器・ドライバの結果からの類推

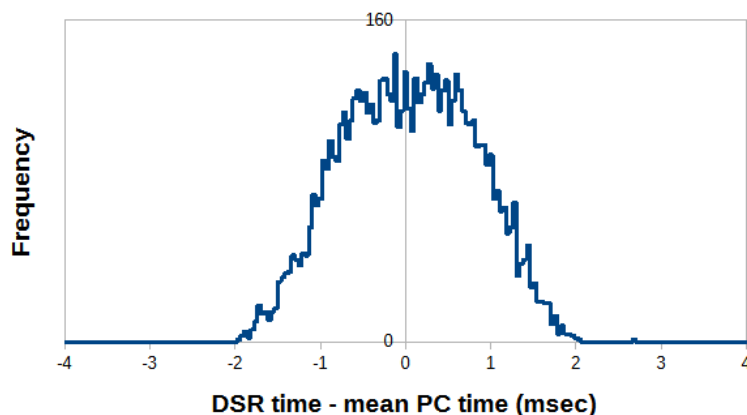


図 13 Garmin18xlv を SABRENT 銘変換ケーブルに接続した場合の DSR 信号検出時刻のばらつき

左： 測定値 右：概念図

図 9 と同様に、DSR 信号検出の時刻のばらつきを調べた。その結果を図 12 に示す。正規分布に近い形となっているが、幅は $\pm 2\text{msec}$ と大きな値である。また、一側の値は -2.0msec を超えることはない。－の値は平均より速い時間に検出されたことを示している。前出の図 11 では、より精度の高い変換器およびドライバを用いた測定よりも、この変換器の場合には約 2.0msec 遅れたところに変化の中央値が見られることから、1PPS 信号の立ち上がりと検出時刻の立ち上がりは、ほぼ同時であろうと考えられる。このように、変換器やドライバが異なると、ばらつきの幅は変化するが、他には共通する点も多く見られることから、PC における信号処理の手順はほぼ同様であると考えることができる。

4. HACSTIP の時刻補正機能

(1) 時刻補正機能の設計

HACSTIP の時刻補正の仕組みを図 14 に示す。

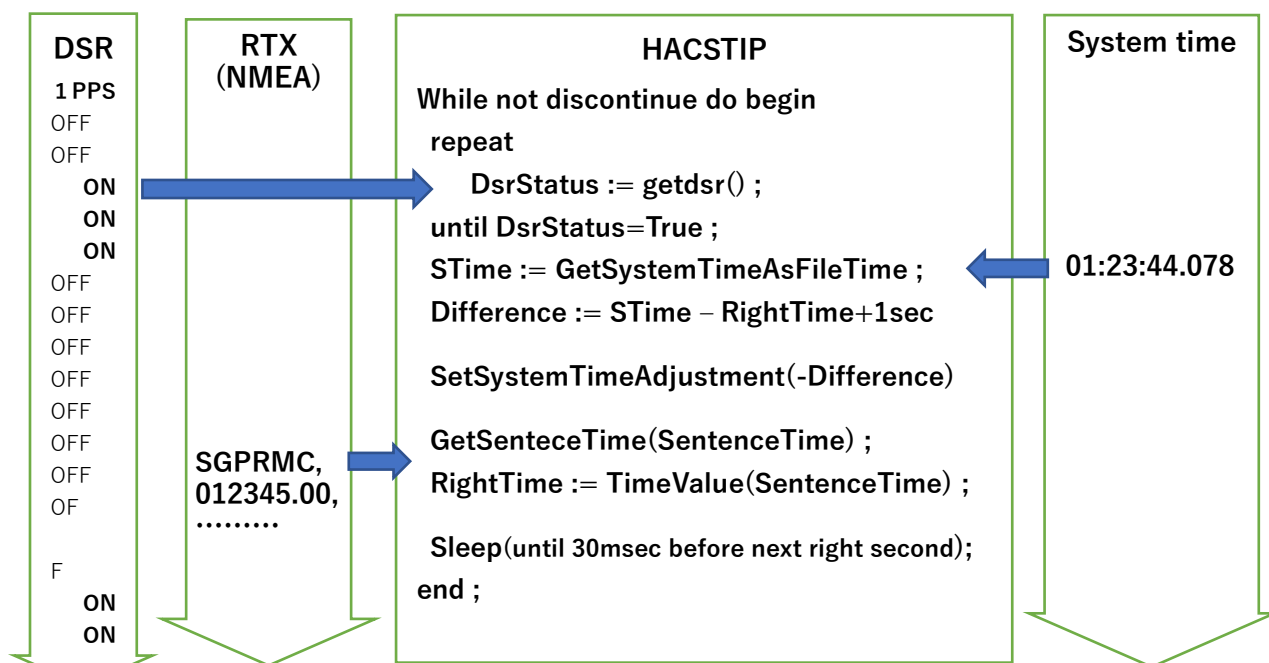


図 14 HACSTIP の時刻補正機能の概要

Free PASCAL の書式で表したものの。簡略化するために、大まかな仕組みのみ示した。関数名も、わかりやすさを優先して、実際のものとは変えてあり、エラー処理も表現されておらず、数値の単位も省略してある。RMC センテンスは 1PPS の少し後から送信されるので、そこから得られる時刻値に 1sec を加えた値と System time の値とを比較して補正を行う。

表3のiの値は、この repeat – getdsr() - until 文の実行回数である。FTDI ドライバの元では1回の処理が $3\sim 5\mu\text{sec}$ で実行されるが、精度は前項に記したように、精度はドライバの検出からデータセットまでの時間に左右される。また、RithTime を得るまでの処理時間が1秒を超えてしまうと、上記の repeat until 文の開始が次の正秒に間に合わない。そのため、PC の能力や環境によっては、30msec という数値も変更する必要があり、HACSTIP はユーザーが値を変更することができるようになっている。(実際には 30msec ではなく、970 (=1000-30) のような値で設定する。) また、もし間に合わないことがあっても、その場合には時刻補正を行わないようにし、全体の補正に影響を出さないようにしている。また、図10に見られるように、DsrHolding 値の読み取りの遅延によると思われるスパイク状の異常値がときおり発生するが、これについては実験結果の統計からばらつきの大きい測定値の $3\sigma \approx 0.25\text{msec}$ を基準値として、それより大きい値を異常値の候補として扱い、異常値候補が単独の場合には補正を行わない。2回以上続いたときは、何らかの要因でシステムタイムが変化あるいは変更されたものと判断して、補正を行う。これにより、異常値に影響されることなく、安定した補正を実現している。

補正は各秒ごとに行われる。DSR 信号の変化を UTC に等しいと仮定して、それとシステムタイムとの差に一定の割合を乗じた値(係数)を、符号を逆にして次の秒までにその値だけシステムタイムを「少しずつ」進めるあるいは遅らせるような処理を行う。

(2) 補正量の適正化

システムタイムの進み遅れが最も適切に補正されるような係数を探した。図15に、係数と補正結果の分布を示す。実験に用いたPCのシステムタイムは進みがちであるので、係数が0.25と小さいときには補正が十分に行われない。係数が1.0よりも少し高い値とすることにより、ほぼ左右が対象となる分布を得ることができた。HACSTIP では、係数として1.15を用いている。図16には補正の時間的な変化の様子を示す。

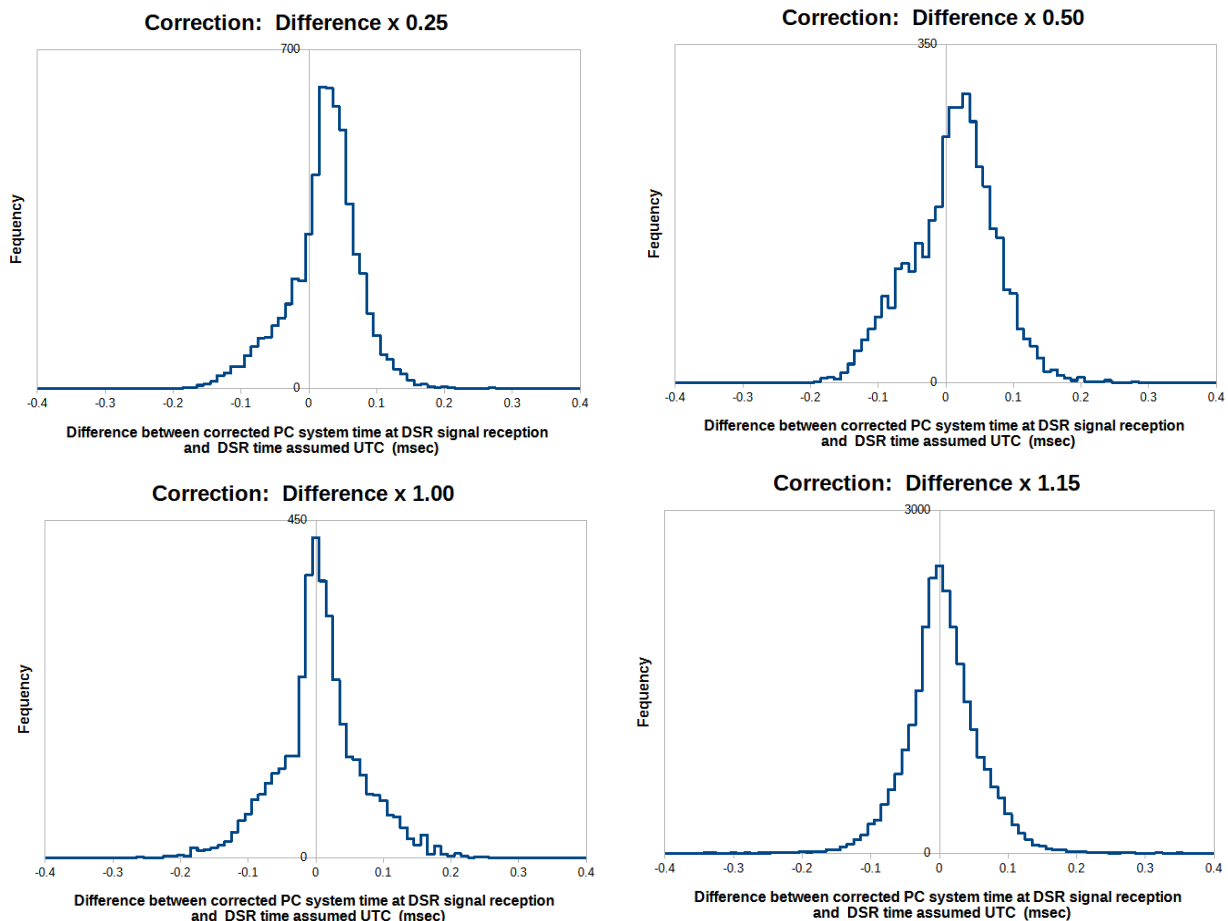


図15 DSR 信号受信を UTC 正秒と仮定したときの時刻値（補正後）の分布

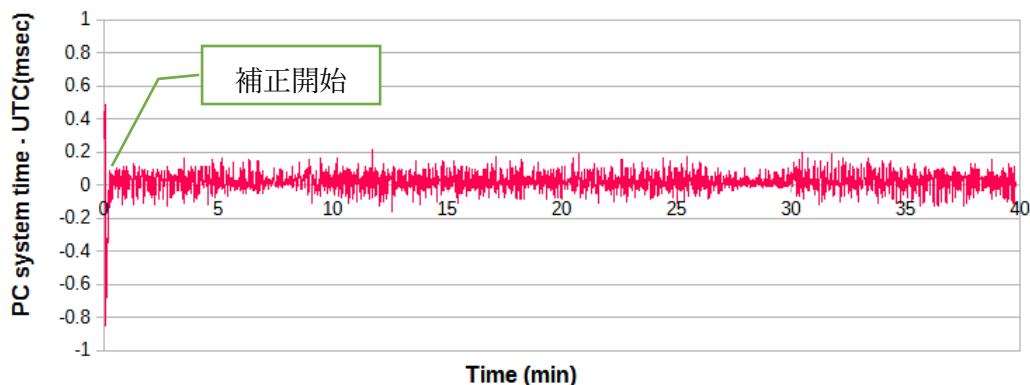


図 16 GPS 受信機からの 1PPS 取得時に PC システム時刻を読み込んだ値と 1PPS (UTC) 信号との差
(改良後 GetSystemTimeAsFileTime および SetSystemTimeAdjustment 関数を用いた結果)

5. 1PPS の LED 光を用いた時刻補正の精度向上への貢献

ビデオカメラが内蔵する基準発信器はきわめて安定した周期で信号を出力しており、それに基づいて安定度の高いフレームレートでの映像データ出力が行われている。そのビデオに GPS の出力する UTC に精密に同期した 1PPS の LED 光を記録して、LED 光が立ち上がる瞬間のフレームに対するタイミングを調べることで、キャプチャソフトウェアがフレーム画面に記録したタイムスタンプの値を良好に補正することができ、それを用いて現象時刻を精度よく求めることができる。

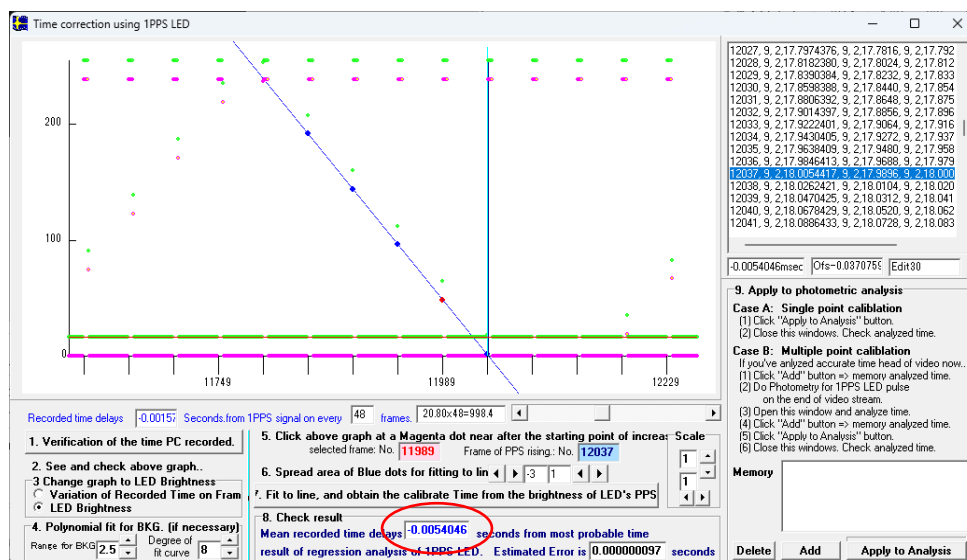


図 17 Limovie の LED 時刻補正機能 SharpCap Timing Analysis

図 17 は、LED 光の立ち上がりとフレームのタイミングを求めているようすである。X 軸は時間（フレーム番号）を表し、このケースの場合のひと目盛りは 48 フレーム（約 1 秒）である。青色の斜め線（検量線）上に並ぶ点がフレームに記録された LED の明るさであり、検量線が X 軸と交わったところ (No. 12037 フレームの露光終了時刻) が LED 光の立ち上がりの瞬間、すなわち正秒であることを示している。SharpCap がフレームに記録した時刻について、ビデオ全体にわたって調べ、その平均の進み方を元に計算したこのフレームの露光終了時刻の値を LED 光の正秒と比較して、どの程度の補正が必要かを求めた結果が赤い円で示された数値である。もし、PC システムタイムが UTC に完全に同期されていれば、この補正量はどの立ち上がり位置で測っても常に一定の値となるはずである。したがって、この値の変化を見れば、システム時計の補正が良好に行われているかどうかを知ることができる。時刻取得関数による補正結果の比較を図 18 に示す。

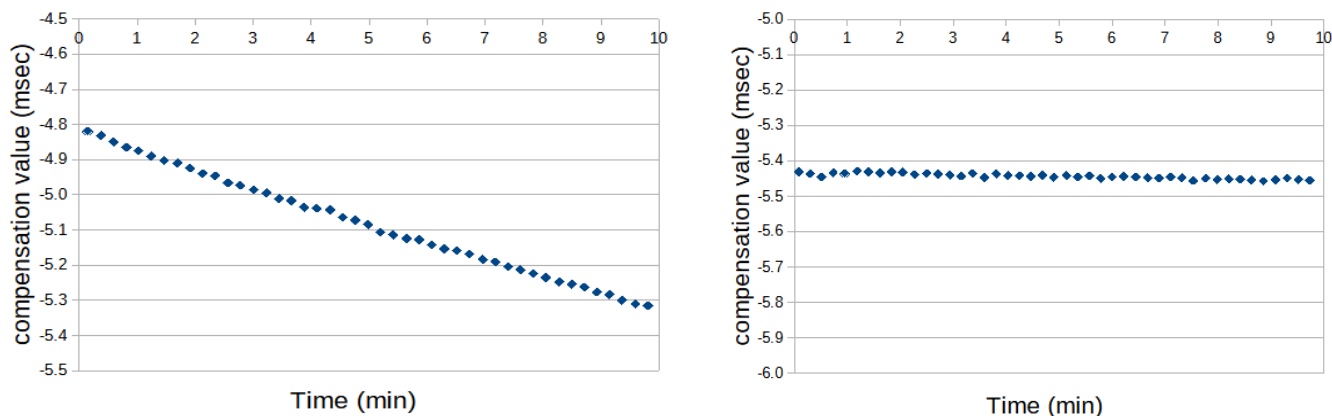


図 18 LED 光による補正值の変化 左：GetSystemTime 関数 右：GetSystemTimeAsFileTime 関数

従来、GetSystemTime 関数を用いた従来の場合（図 18 左）には、補正された PC の時刻を確実に UTC と同期させることができなかったことから、掩蔽現象の記録中には GPS による時刻補正を OFF にして、PC システムタイムの進み遅れが一定速度で起きていることを仮定して時刻の進み方の平均を求め、その上で「LED 光による時刻補正」をせざるを得なかった。時計の進み遅れについては、現象の前後に記録した 1PPS の LED 光により調べることににより行われていた。それに対して、GetSystemTimeAsFileTime 関数による方法（図 18 右）では、録画期間を通して PC システムタイムと UTC が良好に同期することから GPS による PC 時計の補正を常時 ON にした方が精密な観測が可能である。また、LED が十分な明るさで録画できれば、観測の前後のうちどちらか一方での LED 記録で従来以上の精度での観測が可能となる。

6. DSR 経由で 1PPS を出力しない GPS 受信機(VK172 など) から得られる時刻の精度

VK172 は、小さく持ち運びに便利なこと、USB ケーブル 1 本で PC につなげればそのまま使用できること、たいへん安価であること、LED を内蔵していることから、それを筒先から入射することで、精密な時刻補正ができること、など、優れた点も多い受信機である。ただし、DSR ピンからの 1PPS の出力はなく、それを利用した精密な PC システムタイムの補正は充分ではない。また、このように USB に直接つなげるように設計された市販の受信機は全て、DSR についての機能を持っていない。

そのような場合に、時刻管理ソフトウェアは、受信機が発信する NMEA センテンスの出力を検知して、それから何ミリ秒前に正秒を迎えていたのかをあらかじめ調べておいて、その遅延を含めて計算して時刻補正を行う。それでは、NMEA センテンスは、どのようなタイミングと精度で発信されているのだろうか。

新バージョンの HACSTIP での測定結果を図 19 に示す。

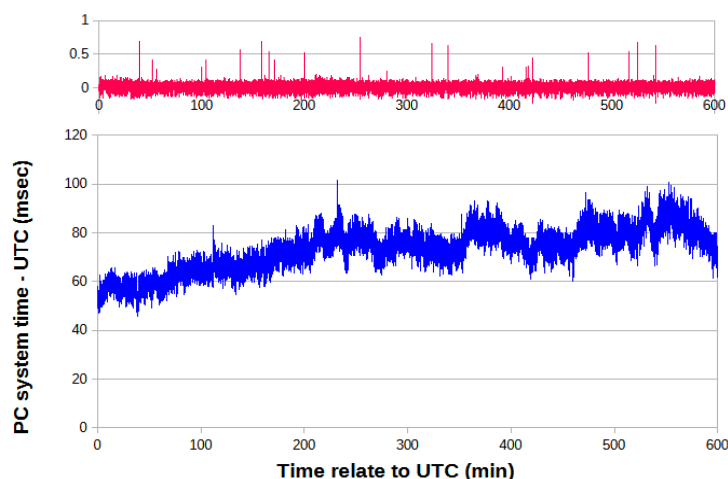


図 19 DSR の 1PPS 信号で補正した時刻（上）と VK172 の NMEA センテンス出力で補正した時刻（下）

従来、NMEA センテンスは正秒に対して 80msec 遅れなど、一定の値として扱われることが多かった。しかしこの測定からは、50msec から 100msec の間で大きく変化していることが明らかになった。したがって、VK172 をはじめとする「USB のみで接続するような既製品の受信機」では PC 時刻を正確には UTC に同期できない。しかし、Limovie の SharpCap Timing Analysis において、 $\pm 0.5\text{sec}$ 未満の時刻値を与えて LED による補正を行うのであれば、そのための時刻取得手段としては有効である。

その場合、現象記録中に補正を常時 ON にすることは、そのために得られる観測精度が低下することから避けなければならない。そこで、新バージョンの HACSTIP は、DSR の使えない受信機の場合には One time correction のみの補正とし、常時補正はできないように改良した。取得時刻の変化が図 19 のようであるなら、誰も常時補正に用いたいとは思わないであろう。

7. GPS 時刻管理ソフトウェア Satk の動作のようす

Satk (さとくん) は 2000 年 (西暦) に故 瀬戸口貴司氏により開発された GPS 利用の PC 時刻管理ソフトウェアである*1。GPS からの 1PPS 信号を RS232C の DSR ピンにつないでソフトウェア側から利用して、高精度の時刻管理を可能としている。それ以降、類似のソフトウェアが登場するようになり、Satk も含めて掩蔽・星食観測に用いられてきた。特に近年、天体用 CMOS カメラが観測に用いられるようになってからは、このようなソフトウェアは必須の存在となってきていた。Satk はリリースから今日まで、多くの掩蔽観測者により利用されてきているが、その登場時期には SetSystemTimeAsFileTime のような精密な関数が動作する OS 環境は一般化しておらず、そのため従来の時刻取得関数によってつくられたソフトウェアである。

ここでは、Satk の動作シーケンスを、新たな関数を使ったソフトウェアを使って調べてみる。

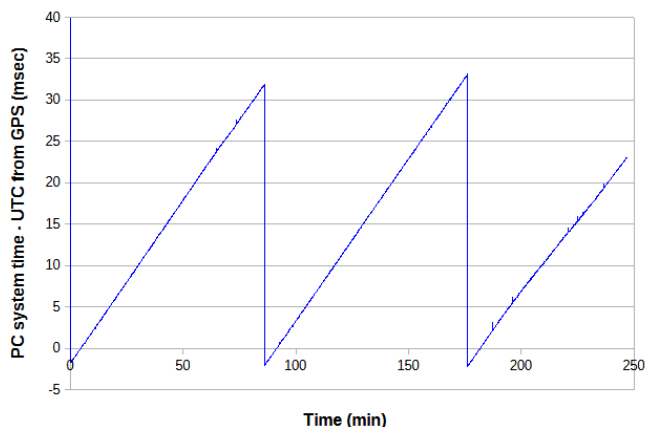


図 20 Satk の時刻補正のようす

Satk を「時刻校正をする」で動作させ、そのときの PC システム時計の時刻推移を HACSTIP の高精度時刻読み取りにより得たものである。一定の割合で PC のシステム時計が「進んで」いく様子を見ることができる。この PC は 1 時間で 20msec ほど UTC に対して進む傾向がある。(図 20)

Satk は、時刻を継続して監視しており、PC システム時刻と UTC の差が 30msec (あるいは 32msec) を超えると PC システム時刻を UTC に合わせて修正するように設計されていることがグラフから読み取れる。

Satk を掩蔽観測に用いるときは、たとえば現象の 10 分前に Satk を起動させる。設定で「時刻校正をする」にしてから動作を続け、そのまま観測を行えばよい。PC システムタイムの進み遅れの状況にもよるが、起動から 1 時間またはそれ以上が経過しなければ補正動作は行われないので、急な時刻補正を避けなければならない Limovie の SharpCap Timing Analysis でも問題は発生しないだろう。

いずれにせよ、様々な受信機に対応して確実な動作が期待できることから、現在でも十分活用できるソフトウェアである。HACSTIP は性能を実現させるために 64 ビット Windows でしか動作しないため、32 ビット Windows の PC では動作しない。その場合、DSR 信号を利用できるソフトウェアは Satk のみであり、時刻管理ソフトウェアの基本形として、これからも利用をお勧めしたい。なお、掩蔽観測に利用する場合には、現象前後の LED 光の録画は必須である。

8. GPS-Clock の動作のようす

これまで、動作シーケンスについての解析を試みているが、どの程度の閾値ならば動作するのもも含めて、十分な検証ができていない状態である。少なくとも、数秒など、ある程度大きな変化があったときには時刻補正が行われるが、数十ミリ秒などのずれに対しては補正が行われよう。今後も検証を続けていきたいと考えているところである。

9. さくら時計の動作のようす

さくら時計は、NTP（Network Time Protocol）を利用して PC システムタイムの補正を行うソフトウェアである。フリーソフトウェアであることや、操作が容易で信頼性が高いことなどから、広く活用されている。さくら時計により時刻補正をおこない、HACSTIP で DSR との時刻差を調べた実験の結果を図 21 に示す。補正結果はほぼ $\pm 5\text{msec}$ 以内に収まっているが、時には 30msec に及ぶ誤差が発生する。ネットワーク回線の状況によっても補正精度が左右されることが考えられ、常にこの実験通りとなるかどうかはわからないが、掩蔽観測の現象記録前に $\pm 0.5\text{sec}$ 未満で時刻を合わせるのに十分活用できる。なお、図で示すような連続的な時刻補正は掩蔽観測には好ましくない。必ず「起動前にオンラインにする」や「常駐する」のチェックを外し、現象前に一度、補正の閾値を 1 ミリ秒にした状態で「オンライン」にする、という形で 1 回の補正として用いるようにすること。また、さくら時計を使う場合は、現象の前後に 1PPS の LED 光を記録する必要がある。

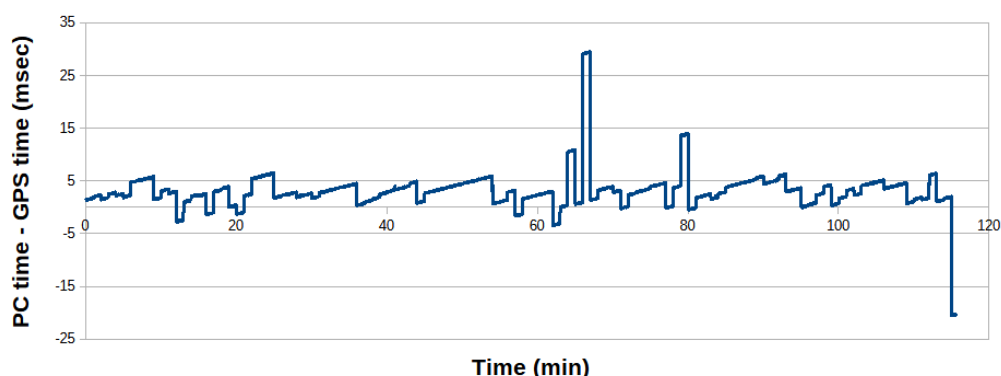


図 21 さくら時計による時刻補正の様子

10. まとめ

以上、高精度の関数を活用することで新たに明らかになってきたことがらについてまとめてみた。今後、より高精度な観測時刻を得ることが求められてくると予想するが、そのときに機材、ソフトウェア、観測技術のそれぞれに新たな改良や工夫が求められてくるものと考えられる。GPS 受信機については、新たな機材の工夫も試みているので、月による星食の観測への応用も含めて、別の機会に紹介したいと考えている

謝 辞

渡辺裕之氏には、新たな受信機についてご紹介やご教示をいただくとともに、HACSTIP の試作版についても長時間にわたるテストを実施していただいた。吉田秀敏氏には、新バージョンを観測に活用していただき、そこから得た貴重な情報を寄せていただいた。渡部勇人氏をはじめ、Didymos 観測チームの皆様には、ソフトウェアの試用および動作速度についての試験をしていただいた。深く感謝申し上げます。

参考資料・文献

1. Satk ドキュメント (瀬戸口貴司, 2005)

<http://www2.synapse.ne.jp/haya/ghsod/Satk.html>

2. PC の GPS 時刻合わせ ハードウェアおよびソフトウェアの例 (渡辺裕之, 2019)

<http://astro-limovie.info/limovie/cmos/GPSCorrectionSoftware.pdf>

3. HACSTIP-GPS 使用説明書 (USB 接続機種 簡単操作編) (宮下和久, 2022)

http://astro-limovie.info/limovie/cmos/HACSTIP_Users_Manual_for_USB_%20direct_connect_receivers-Rev03.pdf

[RS232C 関係]

4. 実験レポート 6:各種 USB-シリアル変換ケーブルで動作する通信速度の調査 (秋山製作所, 2018)

http://www7b.biglobe.ne.jp/~akiyama_manufacturing/akmrsm100/experiment06_20180702.pdf

5. Lazarus 用 RS-232C 通信コンポーネント (千々岩 幸治, 2018)

6. 環境計測に用いられる OS のリアルタイム性の比較 -RTLlinux の導入- (八木英二, 2003)

http://gion.kpu.ac.jp/2003abstract/e_yagi.pdf

7. Delphi Programming / Object Pascal [Mr. Xray] (齋藤 誠, 2019)

http://mrxray.on.coocan.jp/Delphi/Others/TDateTime_SystemTime.htm

8. 高性能プログラミング技法の基礎 (1) (埴 敏博, 2016)

<https://www.cspp.cc.u-tokyo.ac.jp/hanawa/class/spc2016a/sp20161018.pdf>

[Microsoft 社 技術文書]

9. sysinfoapi.h ヘッダー

<https://learn.microsoft.com/ja-jp/windows/win32/api/sysinfoapi/>

10. GetSystemTimeAsFileTime 関数 (sysinfoapi.h)

<https://learn.microsoft.com/ja-jp/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsystemtimeasfiletime>

11. GetSystemTime 関数 (sysinfoapi.h)

<https://learn.microsoft.com/ja-jp/windows/win32/api/sysinfoapi/nf-sysinfoapi-getsystemtime>

12. SerialPort.DsrHolding プロパティ

<https://learn.microsoft.com/ja-jp/dotnet/api/system.io.ports.serialport.dsrholding?view=dotnet-plat-ext-7.0>