RighTime[tm]
A Real Time Clock Correcting Program For MS–DOS–Based Computer Systems

G. Thomas Becker
Air System Technologies, Inc.
14232 Marsh Lane, Suite 339
Dallas, Texas 75234–3899

Abstract

A computer program is described which effectively eliminates the misgivings of the DOS system clock in PC/AT–class computers. RighTime is a small, sophisticated memory–resident program that automatically corrects both the DOS system clock and the hardware "CMOS" real time clock (RTC) in real time. RighTime learns what corrections are required without operator interaction beyond the occasional accurate time set. Both warm (power on) and cool (power off) errors are corrected, usually yielding better than one part per million accuracy in the typical desktop computer with no additional hardware, and RighTime increases the system clock resolution from approximately 0.0549 second to 0.01 second. Program tools are also available which allow visualization of RighTime's actions, verification of its performance, display of its history log, and which provide data for graphing of the system clock behavior. The program has found application in a wide variety of industries, including astronomy, satellite tracking, communications, broadcasting, transportation, public utilities, manufacturing, medicine and the military.

Introduction

Most MS–DOS, PC–DOS and DRDOS users have long ago learned to live with, and generally regard as inadequate, the system time–of–day clock that is a standard component of these operating systems. The typical DOS–based computer system clock exhibits inaccuracies that can range from a few seconds to several minutes per day, and the system can lose track of days at a time (on Friday, leave a DOS–based computer running at the office and go home for the weekend; when you return to the office Monday morning, the machine will very likely tell you it's Saturday). These errors are the result of a combination of compromises at several steps in the IBM PC design process, *circa* 1980. The consequences remain with users to this day.

No autonomous RTC hardware was implemented in the original IBM PC announced in August, 1981. Instead, software (partially in ROM–based BIOS firmware and partially in RAM–resident DOS) counted regular interrupts which were generated by an interval timer. DOS converted the resulting "tick" count to conventional expressions of time of day when application software called for it, and, if during a request for the time DOS determined that midnight had passed, it

incremented the system date. Whenever the system was booted, the date and time was initialized to 1980/01/01 00:00:00; the user was expected to set both at each boot.

With the introduction of the PC/AT in August, 1984, a hardware clock system — a Motorola MC146818 CMOS RTC and some associated circuitry — was included. The part was powered by a battery when the system was not operating so it would maintain date and time continually. Released at the same time, DOS Version 3 automatically read the CMOS RTC clock date and time at system boot and set its date and tick counts to match; aside from that initialization at boot, the hardware clock was not used. When the DOS date or time was changed by the user, the change was not reflected in the CMOS RTC clock; it needed a separate setup utility program. Later DOS versions set the CMOS RTC clock at the same time the DOS clock was set, but the CMOS RTC clock time was still read, and is still today, only at boot.

No other changes have been made to the time–of–day clock mechanisms of the AT–class PC–compatible computer since 1984, except as have been applied to the CMOS RTC clock hardware itself. (Dallas Semiconductor and Motorola now produce several compatible lithium–powered clock modules and Intel and other semiconductor producers have incorporated the RTC logic within large–scale integrated circuits that support the current microprocessors.) The hardware manufacturers and software publishers moved on as if whatever few difficulties that existed were solved with the advent of the PC/AT. The problems were neither solved nor few.

The Problems of The DOS Clock

The interval timer that DOS uses to generate the regular tick that it counts is driven by an uncalibrated, unconditioned and usually non–adjustable 1.193 MHz source (1.193 MHz is one quarter of the original PC system clock frequency of 4.77 MHz, and it is also one third of the NTSC television standard color burst frequency, 3.579 MHz). The interval timer is programmed to divide that frequency by 65,536 to produce an interrupt rate of approximately 18.206 ticks per second which determines the resolution of the standard DOS clock, approximately 54.925 milliseconds. DOS allows for the expression of time in 0.01 second increments, but it cannot internally maintain that resolution nor represent decimal times exactly due to the tick resolution.

DOS depends upon the cooperation of all software that runs on the machine to allow sufficient time for each interval timer tick interrupt to be counted. This is not always possible, so some interrupts are missed and are not counted. On a computer whose interrupt system is heavily loaded, the accumulation of lost interrupts makes the DOS clock appear to run slowly.

The tick interrupt is normally intercepted by a routine in the ROM BIOS which increments a 32–bit tick count in RAM and determines if a value that represents approximately 24 hours has been exceeded (the tick duration makes 24 hours indeterminable exactly). If so, the tick count is set to zero and a single bit flag, representing the passing of midnight, is set and provided to DOS when it next asks for the tick count. These DOS requests only occur when the system is in need of the current time of day. At that time, if the bit is set, DOS increments the date. If more than 24 hours have elapsed between two requests for time of day, DOS is unable to recognize that more than one midnight has passed. If the user displays the date after a system has been running, but idle for more than a day, DOS will indicate that the date is one day after the last day the machine was used.

The interval timer has found itself reprogrammed by a number of applications, usually games that require dynamic video presentations. In these programs, the interrupt that the interval timer yields is used to refresh the video screen at a high rate. At the termination of the game, the DOS clock is rarely where it should be. Some of the more responsible programs make an attempt at resetting the DOS clock to compensate for the game session duration, but others leave it in disarray. If the game increased the tick rate and left the intercepting BIOS interrupt "hook" in place, the DOS clock will have advanced greatly; in the worst examples, the rate is left accelerated as well.

The Problems of The CMOS RTC

The CMOS RTC clock resolves only to whole seconds. When DOS reads it at boot, one second of ambiguity is set in the DOS clock even if the CMOS RTC clock is accurate. Conversely, when the DOS clock is set, DOS transfers whole seconds to the CMOS RTC clock but makes no attempt to deal with the fractional part of the seconds, nor does it synchronize the seconds transition, so even if the DOS clock is set accurately, the CMOS RTC clock won't be.

The CMOS RTC clock is usually paced by a 32.768 kHz quartz crystal which is loaded by a pair of capacitors. The crystal is often an inexpensive watch–type device, characterized for operation at room temperature (typically 25 degrees C). At temperatures either above or below the characterized temperature, the crystal's resonance change will slow the clock. The loading capacitors are generally not temperature compensated and add to the detuning. Except in the case of the modern CMOS RTC modules mentioned earlier, no provision is made to allow trimming the crystal frequency. Dallas Semiconductor RTC modules are internally trimmed at the factory to +20 seconds per month at room temperature, anticipating an environment that will slow the clock; in use, the computer system internal temperature is widely varying, ranging from perhaps 10 degrees C when the system is not powered to 55 degrees C when it is. Dallas Semiconductor's data indicates that the part can be expected to run as much as 2.5 minutes per month slow at these temperatures.

The CMOS RTC clock oscillator is powered by the system's +5 volt supply when the machine is powered and by a lower voltage when it is on battery backup. Typically, an additional six parts per million error can be expected when the clock is running on battery power.

The CMOS RTC clock part provides an option that will automatically advance or retard the time at 0200 on the appropriate Sunday morning when Daylight Savings Time and Standard Time, respectively, start in the USA. This feature is usually not used, since no mechanism to invoke it exists in DOS. DOS does, nevertheless, include a bit in its internal time data structure that indicates whether this automatic time change feature is enabled or not (many documents incorrectly state that the bit indicates that the time is Daylight Savings Time). The rule was changed in 1986 and many computers — even current models — contain parts that still adopt the pre–1986 rule. Even if the feature is enabled, many machines will change on the wrong Sunday morning in April (the last rather than the first). Dallas Semiconductor's DS1287 RTC module was introduced with the currently correct rule. Motorola corrected its part with the announcement of the *MCCS146818B1M RTC module in* 1991.

The Solution to the Problems

RighTime approaches the solution to these problems by making best use of the better qualities of each of the two clocks. Two foremost qualities of the CMOS RTC are its relative stability and its autonomy. The DOS clock has a high resolution interval timer available to it, although its availability is conditional, subject to loss of power and to software that commandeers it.

With these resources, RighTime does four fundamental things:

1. RighTime slaves the DOS clock to the CMOS RTC, frequently referring the DOS date, tick count and interval timer count to the CMOS RTC date and time;
2. RighTime operates the interval timer in a disciplined mode that allows deriving a higher resolution than standard while maintaining the standard tick rate and count for compatibility;
3. RighTime maintains accuracy by regularly calculating and applying corrections to the DOS clock and adjustments to the CMOS RTC; and
4. RighTime intercepts and monitors time set commands to learn and refine the CMOS RTC correction rate.

RighTime is loaded each time DOS is booted, makes itself resident in system memory, and hooks a number of system interrupts. (A hook is a logical tap in the execution chain of some specific event or class of events.) For its time-keeping tasks, RighTime hooks an interrupt from the CMOS RTC, one from the interval timer, and several that convey the time set and read commands from an application program or the user to DOS. Other interrupts are intercepted to disallow functions that could otherwise change resources that RighTime must exclusively control. Since DOS is not ordinarily a multitasking operating system, several other interrupts are hooked to determine when system activities are momentarily idle. These logical pauses are often very brief, but occur frequently and allow RighTime to do its manipulations as a transparent background function without noticeably affecting work that is ongoing in the foreground.

The CMOS RTC is programmed to produce an interrupt once per second, immediately after its internal seconds update. Every four seconds, RighTime reads the CMOS RTC date and time, calculates the equivalent DOS clock values, and sets the tick count and interval timer count and mode accordingly. This regular update prevents the DOS clock from wandering far from the CMOS RTC time and corrects whatever ills might come from a game or other application that reprograms the interval timer. Between these periodic updates, the DOS tick count is incremented by the BIOS interval timer interrupt handler — just as it is without RighTime — and is available to those programs that use the tick count directly.

RighTime augments the routines that handle the conversions of tick count to time-of-day and time-of-day to tick count with its own routines. These routines process the Get DOS Time and Set DOS Time functions, respectively, and provide resolution that far exceeds the 0.01 second resolution of the DOS clock data structure. Any program that gets the time via the DOS calls will benefit from the additional resolution that RighTime provides.

Although the CMOS RTC is relatively stable, its rate is rarely highly accurate. Since, under RighTime, the DOS clock mimics the CMOS RTC, the DOS clock would exhibit the same rate error unless corrected. RighTime handles this by incorporating a calculated correcting offset in

each of the DOS clock updates. The result is a deliberately increasing divergence of the two clocks. Assuming that the corrected DOS clock rate is accurate, the CMOS RTC must be occasionally adjusted to match the DOS clock. The CMOS RTC is advanced or retarded when the difference between the two clocks has reached one second; the DOS clock offset is adjusted accordingly. If the DOS clock is set to the correct time, and the CMOS RTC rate correction is proper, the result is a DOS clock whose expression is within 0.01 second of the correct time and a CMOS RTC that is within one second of the correct time. Actually, the calculated DOS clock offset is bipolar, so the CMOS RTC is never more than +/-0.5 second from the correct time.

Since the CMOS RTC rate can vary with changes in temperature and voltage, the theoretically perfect correcting algorithm would account for both in appropriate complex terms. In practice, two rates are maintained by RighTime, and suffice; one represents the system power–off state and the other represents power–on. We refer to them as "cool" and "warm", though, since most users seem intuitively familiar with temperature effects. RighTime applies a single calculated adjustment when it is loaded at system boot that corrects for CMOS RTC error that accumulated while the system was in the power–off state; if the program has not been running for more than 30 minutes, the cool correction rate is employed, otherwise the warm correction rate is applied. With that exception, the warm correction rate is used in each DOS clock update.

The CMOS RTC rate correction values are learned. RighTime assumes that whenever the DOS clock is set, it is set accurately. At the moment of set, then, the existing DOS clock error is a function of the CMOS RTC rate error, the current cool and warm correction rates, the time elapsed since the last accurate time set, and the ratio of warm and cool operation in that period. The correction rates are refined with each time set, yielding decreasing error as they close on the "ideal" values. These values, and others that are necessary for the process, are stored in a suitable place within the machine hardware or in a small dedicated file.

For those users who want one less clock to change twice each year, RighTime offers support for the American Daylight/Standard time change feature of the CMOS RTC. Since the change itself is handled by the hardware, the program does not need to be running when the change needs to occur. The user is cautioned, though, that the change might not occur on the prescribed day in April due to an outdated hardware component.

Although RighTime can usually be used effectively with the defaults, the program accepts a wealth of options and user–provided parameters that can detail the function to a specific machine environment, and a logical interface is included that allows the time setting functions to be automated. A system can be easily designed that will continually adjust itself to a slowly changing operational environment as might be encountered with seasonal temperature changes at a remote data acquisition site.

Conclusion

RighTime is an effective software solution to the poor time–keeping performance of the PC/AT–compatible, DOS–based computer system. The cesium beam and hydrogen maser clock industries will remain unchallenged by your computer, but it should be easy to achieve error rates of 0.5 second per week or better. We regularly hear reports of one tenth of that error rate from stable systems, but these represent the best that can be expected in the current form of the program.

The current version (v2.46) requires only 6.5K of RAM, and the program can be loaded into high memory.

RighTime is a commercial software product that is currently distributed via shareware. An evaluation copy is available electronically on the Air System Technologies BBS, 214/869–2780 (300–9600 bps, 8N1), free of charge. The evaluation program is fully functional and may be used for up to 30 days. Usage beyond that period requires payment of a registration fee.

Development of the RighTime process is ongoing. We anticipate reliable accuracies on most hardware to exceed 0.001 second, allowing DOS clock time to be expressed to the millisecond in a future revision. An OS/2 version is in planning, with a summer 1993 anticipated delivery.

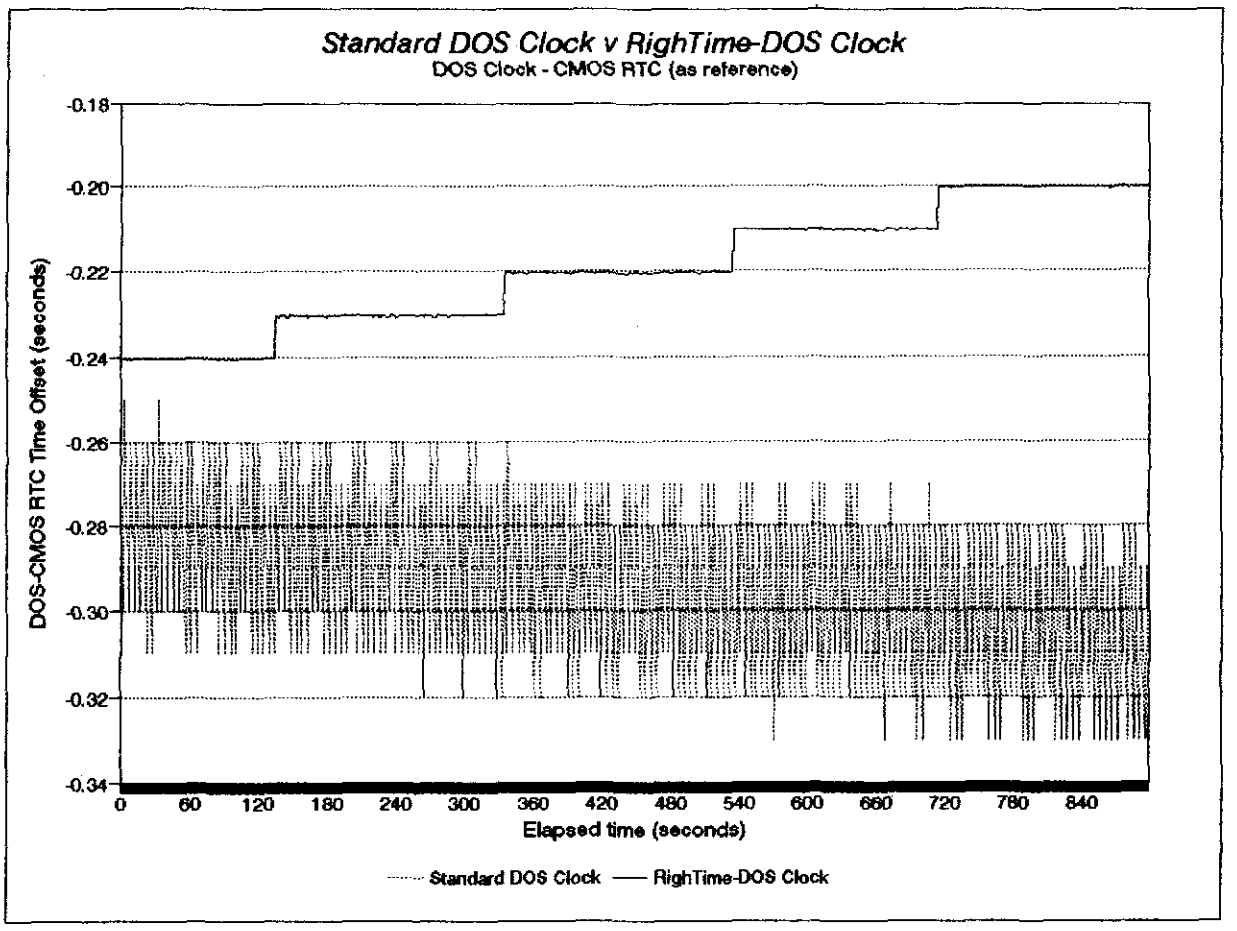

Figure 1. Comparison of the uncorrected, standard-resolution DOS clock to the RighTime-corrected DOS clock. The data is the difference between the indicated DOS clock time and the CMOS RTC time, measured at the moment of the CMOS RTC seconds update. The broad swath of the standard DOS clock is due to its inability to express decimal time values as a result of its poor resolution. The general downward trend results from the combined rate error of the DOS clock and the CMOS RTC. The comparatively fine, stepped RighTime-corrected DOS clock data shows the tight regulation (typically well within 1 ms) and the deliberately increasing applied time offset that compensates for the CMOS RTC rate error. These data were taken from a machine whose CMOS RTC runs about four seconds per day slow.

```
C:\RT2>rightime

RighTime: Indicated DOS clock date and time is 1993/01/05 20:54:31.91.
RighTime: Warm correction rate is -0.62 second per day.
RighTime: Cool correction is defeated. All adjustments use warm rate.
RighTime: Current applied DOS-CMOS RTC offset is -0.03 second.
RighTime: Last DOS clock correction was 1.94 seconds ago.
RighTime: Last CMOS RTC adjustment was 0.07 hour ago.
RighTime: Last timeset was 1.94 hours ago.
RighTime: Stack A headroom is 146 bytes; Stack space used is 62 bytes.
          Stack D headroom is 158 bytes; Stack space used is 50 bytes.
RighTime: /?=Help; Version 2.46; DEVELOPMENTAL PROGRAM: DISTRIBUTION PROHIBITED
RighTime: Copyright 1991-92 GTBecker, Dallas 214/402-9660. All Rights Reserved.
RighTime: Selftest passed.
RighTime: ** Already resident. **

C:\RT2>testtime

TestTime: Version 2.46; DEVELOPMENTAL PROGRAM: DISTRIBUTION PROHIBITED
TestTime: Copyright 1991-92 GTBecker, Dallas 214/402-9660. All Rights Reserved.
TestTime: RighTime v2.46 is resident.
TestTime: CMOS RTC mode is normal.
TestTime: "Appl" data is the current DOS-CMOS RTC offset applied by RighTime.
          "Meas" data is the current DOS-CMOS RTC offset measured by TestTime.
TestTime: If = in CMOS data below is flashing, CMOS RTC interrupts are normal.
          Flashing ! indicates the moment of DOS clock correction.
TestTime: Press any key to exit.

TestTime: CMOS=930105:20:55:46 DOS=930105:20:55:46.54 !Appl=-0.03 Meas=-00.0301

C:\RT2>testincr

TestIncr: Version 2.45 Copyright 1991-92 GTBecker, Dallas. All Rights Reserved.

TestIncr: Difference of successive unique DOS clock reads (seconds):
          58.49 - 58.48 = 0.01

TestIncr: There were 500 unique values over 5 seconds, or 100.000 per second.
TestIncr: The average increment over this period was 0.01000 seconds.
```

Figure 2. Screen prints of RighTime (when resident), TestTime, a diagnostic program, and TestIncr, a DOS clock resolution verification program. A log program (not shown) also displays all system time-related activities while RighTime is resident.

QUESTIONS AND ANSWERS

J. Levine: NIST: It should work just as well; you have done a very nice job with it. There are two comments that I was going to make which have nothing to do with your program, rather with people who might not realize two consequences of your program. First of all is that if your clock is fast, then your program will be continuously setting it backwards. That may confuse utilities like MAKE and that is not a bug of the program. That is simply a feature that you just have to be aware of, that you can wind up in which a derivative file is a time that is earlier than is apparent; because the clock was set backwards. The second thing that I wanted to point out is that there are some versions of that real time clock chip which have an incorrect daylight saving time algorithm. They used the old U. S. daylight saving time algorithm rather than the new U. S. daylight saving time algorithm. If you have the older type PC's and you do not call somebody you will be wrong by an hour for a few weeks in April.

G. Becker, Air Systems Technologies, Inc.: Yes, as a matter of fact the Motorola MC146818 was unfortunately modeled; it uses the old rule, it went out of date in 1986. It unfortunately was modeled by many of the VLSI manufacturers for modern machines, so we have machines that were designed in 1990 and manufactured in 1991 that use a 1986 rule. The Motorola chip has been fixed. There is a 146818 MCCS1B, I think that has the new rule in it and the Dallas Semiconductor DS1287 series are also correct. You are quite right if you adopt to use this slash A option on the command line level and you find that the clock has suddenly sprung forward. It will miss the first Sunday, instead will spring forward on the last Sunday of April. It is not the program's fault. It is either an old chip or you can blame your hardware manufacturer for not being up to date. It still is a very common error. I think it was published in Computer Language Magazine, which went to great lengths to publish an algorithm so you could adopt this erroneous rule in C language.

G. Winkler, USNO: I have two questions and one comment. The comment is first. Since there are several computer people here, we have to do something about that awful custom of calling the Gregorian Date a Julian Date. It has been somehow introduced about 20 to 25 years ago to call the day of the year and the year combination the Julian Date. This is of course a Gregorian Date. There is complete confusion about this; there is a modified Julian Date which we discussed before. There is a Julian Date in use for the last 400 years in astronomy. I think you ought to keep these things clearly separated. That is the first thing. Another question is what happens when you have your computer turned off and then is turned on. The rate of your CMOS clock would be different because of the different temperature inside the computer. Does your program correct for that also?

G. Becker: There are actually variables. One is temperature and the other is the power supply voltage. The clock module falls back to usually 3.6 volts (a lithium cell). Sometimes, however it is backed up by four AA cells, or something like that (six volts). In any event, the voltage changes as does temperature. Yes, Right Time developed both warm and cool corrections and will, given enough opportunity, correct adequately for both of those.

G. Winkler: That is wonderful and I am really impressed by that. Now comes another question. I have three operating systems on my computer; DOS, OS2 and Coherent. Of course, I boot up alternately by using the Boot Manager. Is it correct to say that your program will only come into action when we boot using the ALT Executive file on drive C?

G. Becker: Yes, at this moment, it is only in a MS-DOS program. There is an OS2 version coming that will be executionally compatible. You will be able to switch between those two operating systems. The data that one produces can be used by the other. Going into a third, perhaps UNIX or some other operating system, Right Time will not be running. However,

there is a facility so that you can tell Right Time when you come back into either of the operating systems that it will run in, that it has not in fact been comatose. The system still has been running and warm and so it will not mistake that dead period for a cool correction but instead apply the appropriate warm correction to span the lapse of the real time correction.