

Supplement
to the
GPIB-PC User Reference Manual
GPIB-PC Clock Software
April 1986 Edition
Part Number 320028-01

National Instruments Corporation
12109 Technology Boulevard
Austin, Texas 78727-6204
(512) 250-9119

© Copyright 1986 National Instruments Corporation
All Rights Reserved

Notice About Warranties

The DISKETTE on which the GPIB-PC clock software programs are furnished is warranted to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from date of shipment from National Instruments or its authorized dealer. Defects caused by misuse, abuse, or shipment are not covered.

Defective equipment that is subject to this limited warranty and that was purchased directly from National Instruments will be repaired or replaced, at its option, if National Instruments is notified during the warranty period. A Return Material Authorization (RMA) number MUST be obtained from National Instruments before returning any equipment. Shipping costs to National Instruments must be paid by the owner. Shipping costs from National Instruments will be paid by National Instruments. Equipment should be packaged in the original shipping container, if possible, and the RMA number MUST be clearly marked on the outside of the package. The package should be insured during shipment for the amount of the replacement cost of the equipment. Before returning equipment not purchased directly from National Instruments, contact the dealer from which it was purchased for return policies and procedures.

THE GPIB-PC CLOCK SOFTWARE PROGRAMS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. NATIONAL INSTRUMENTS SHALL NOT BE LIABLE OR RESPONSIBLE FOR ANY KIND OF DAMAGES, INCLUDING SPECIAL, INDIRECT, OR CONSEQUENTIAL DAMAGES ARISING OR RESULTING FROM ITS PRODUCTS, THE USE OF ITS PRODUCTS, OR THE MODIFICATION TO ITS PRODUCTS.

Trademarks

GPIB-PC is a trademark of National Instruments.

CP/M is a trademark of Digital Research.

MS-DOS is a trademark of Microsoft Corporation.

GPIB-PC Clock Software

The following is a description of the software provided with the National Instruments GPIB-PC time-of-day clock. In the descriptions below, the term "clock" refers to the time-of-day clock on the GPIB- PC, and the term "system clock" refers to the computer's internal clock. (For computers running CP/M there is no system clock.)

Software Overview

The National Instruments clock software consists of two main parts:

1. a utility program, NIDATE, that allows you to set and retrieve the current day, date, and time from the clock
2. a set of functions to manipulate the clock from BASIC, C, and 8086 assembly language.

For MS-DOS users, there is an additional program, AUTODATE, which will automatically set the system clock from the time-of-day clock each time you boot the system.

Executable Programs

Before continuing, make a working copy of your distribution diskette. Store the distribution diskette in a safe place.

NIDATE - Set or Display the Day, Date, and Time

The NIDATE program allows you to read and set the date, time, and the day of the week. If invoked with no argument, NIDATE returns the current day, date, and time. NIDATE sets the specified date and time and is invoked with an argument of the form: MM/DD/YY HH:MM:SS NIDATE.

The day of the week is computed automatically.

Examples:

1. To read the current setting of the time-of-day clock type:

NIDATE

The response is: **Fri 10/5/84 16:45:23**

2. To set the clock to today's date type "NIDATE", followed by today's date and time. If today were November 1, 1984, 9:14:12 a.m., you would type:

NIDATE 11/1/84 9:14:23

The response is: **Thu 11/1/84 9:14:23**

3. To set the date to November 2, 1984 and leave the time unchanged type:

NIDATE 11/2/84

The response is: **Fri 11/2/84 9:14:56** (33 seconds later)

4. To set the time to 9:15 p.m. exactly, type:

NIDATE 21:15:00

The response is: **Fri 11/2/84 21:15:00**

or

NIDATE 21:15 (seconds default to zero)

The response is: **Fri 11/2/84 21:15:00**

AUTODATE - SET SYSTEM CLOCK (MS-DOS only)

If you are using version 2.00 or later of MS-DOS, you may use the AUTODATE program to automatically set the system date and time from the clock whenever you reboot the system. Do this by copying the program AUTODATE.EXE to your boot area (the diskette you boot DOS from).

Then, in your boot area, create or modify an existing AUTOEXEC.BAT file to include the line AUTODATE.

To do this, use the standard DOS editor EDLIN. Assuming EDLIN.COM is in the boot area on drive A, at the A> prompt enter:

```
EDLIN AUTOEXEC.BAT
*i1
  1:autodate<cr>
  2:<control c>
*e
```

The above sequence will create or add to an existing AUTOEXEC.BAT file the line AUTODATE, which causes the autodate program to be executed whenever the system is booted.

Note: The DOS DATE and TIME commands may still be used to set the system clock; however, they will not affect the time of day clock.

Clock Functions

The remainder of this manual is for users wishing to write software that accesses the clock. The clock functions are simple, independent functions, callable from your program, to manipulate the clock chip. Capability is provided for enabling and setting the frequency of clock interrupts.

The functions are provided in BASIC, C, and 8086 assembly language. Consider them to be generic language implementations. Because there are minor differences between different versions of BASIC, C and 8086 assembly language on the market, the source code provided here is not targeted for any one supplier's implementation of the language. Therefore, minor changes may be required to these source packages to make them conform to any one specific implementation of that language.

BCD Notation

Many functions listed below refer to a structure (a set of memory locations) defined to be of type "time", or TM% in BASIC. Functions using variables of type "time" expect the structure to contain Binary Coded Decimal (BCD) formatted values. In BCD, each four bits of the byte represents a single

decimal digit. For example, the decimal value 14 is represented in BCD in memory as: 0001 0100.

The easiest way to write programs that adhere to the BCD convention is to initialize and print the clock data using the hexadecimal format, then to interpret this data as BCD, or decimal. For example, to set the minutes in the clock to decimal 15, you initialize set.min to hex 15 which generates the appropriate BCD value in memory. Note that only the hex digits 0 through 9 are valid when used this way, since they correspond to the legal decimal digits.

Calling BASIC Language Clock Functions

The following paragraphs describe how to call the BASIC language clock functions from a BASIC language application program.

In the examples, hexadecimal numbers are preceded with a &H. Remarks in the code are delimited using a single quote.

The BASIC functions are in the form of subroutines, and reserve line numbers 10000 through 11000. The examples that follow assume these subroutine line numbers are unchanged. If necessary, you may change the function line numbers provided the change is carried out consistently when using the programming examples.

GPIB-PC2A

If the base I/O address is not hex 02E1 the following changes must be made to CLOCK.BAS:

1. Line 1149 - The value of CCR% is base address + &H9400. For a base I/O address of hex 22E1, CCR% = &HB6E1.
2. Line 1150 - The value of CDR% is base address + &H9800. For a base I/O address of hex 22E1, CDR% = &HBAE1.
3. Line 1177 - Words 5 + 7 must correspond to CCR% and CDR%, respectively, as calculated above.

Table 1 - Clock Function Summary - BASIC Language

Description	Function	Parameter	Syntax
Get the current alarm setting	GETALARM	TM%	GOSUB 10200
Set the alarm	SETALARM	TM%	GOSUB 10300
Get the current clock setting	GETCLOCK	TM%	GOSUB 10400
Set the clock	SETCLOCK	TM%	GOSUB 10500
Set the periodic interrupt frequency	SETFREQ	N%	GOSUB 10600

The argument TM% is a one dimensional array which is indexed using variables declared below. These variables are assumed to have been set before calling the clock functions.

```

DIM TM%(8)
MSEC% = 0      ' TEN THOUSANDTHS OF SECONDS
CDSEC% = 1     ' HUNDREDTHS AND TENTHS OF SECONDS
SEC% = 2       ' SECONDS
MIN% = 3       ' MINUTES
HR% = 4        ' HOURS
DAY% = 5       ' DAY OF THE WEEK
CDATE% = 6     ' DAY OF THE MONTH
MO% = 7        ' MONTH
YR% = 8        ' YEAR
    
```

The argument N% is an integer variable.

These declarations are supplied in the source code beginning at line 10100. Other declarations needed by the functions are also provided between lines 10100 and 10200. Check these to be sure no naming conflict occurs with your application program variable names.

Purpose: Get the current alarm setting

Format: GOSUB 10200 'GETALARM

Remarks: Return the current alarm setting in the array TM% (previously defined).

Each component of TM% contains two BCD digits.

The day of the week is returned as a number from 1 through 7, representing Sunday through Saturday. The YR%, MSEC%, and CDSEC% components are set to zero.

Example:

1. Get the alarm setting and print it. (Note: the data is printed using hex format to convert the BCD-stored values to appropriate ASCII decimal numbers):

(Note: the "TIME" print statement should be one continuous line)

```
GOSUB 10200 'GETALARM
PRINT "DAY: "TM%(DAY%)
PRINT "DATE: "HEX$(TM%(MO%))/"HEX$(TM%(CDATE%))
PRINT "TIME: "HEX$(TM%(HR%))":"HEX$(TM%(MIN%))
      ":"HEX$(TM%(SEC%))".
```

Purpose: Get the current clock setting

Format: GOSUB 10400 'GETCLOCK

Remarks: Return the current time, day of the week, and date in the array TM% (previously defined).

Each component of TM% contains two BCD digits. The MSEC%, CDSEC%, and SEC% components contain the number of seconds into the current minute to an accuracy of 0.1 millisecond: SS.MMUU.

The day of the week is returned as a number from 1 through 7, representing Sunday through Saturday. The YR% component is the year minus 1984.

Example:

1. Get the clock setting and print it. (Note: the data is printed using hexformat to convert the BCD-stored values to appropriate ASCII decimal numbers):

(Note: the "DATE and TIME" print statements should each be one continuous line)

```
GOSUB 10400          'GETCLOCK
PRINT "DAY: "TM%(DAY%)
PRINT "DATE: "HEX$(TM%(MO%))/"HEX$(TM%(CDATE%))
              "/"TM%(YR%)+84
PRINT "TIME: "HEX$(TM%(HR%)):"HEX$(TM%(MIN%))
              ":"HEX$(TM%(SEC%))"."HEX$(TM%(CDSEC%))
              HEX$(TM%(MSEC%))
```

Purpose: Set the alarm

Format: GOSUB 10300 'SETALARM

Remarks: Set the alarm time to that indicated in the array TM% (previously defined).

If the HR%, MIN%, and SEC% fields of TM% are all 0, alarm interrupts are disabled.

When the alarm occurs, an interrupt is generated. You are responsible for setting up the interrupt vector and routine prior to calling SETALARM.

The YR% component of TM% is ignored as are the MSEC% and CDSEC% components. The values passed in for YR%, CDSEC%, and MSEC% in the set variable are ignored, i.e., the alarm resolution is no finer than 1 sec.

Examples:

1. Set the alarm time to 3:15:00 a.m. on June 1. (Note: the use of hex notation equates to assigning BCD values to TM%):

```

TM%(MO%) = &H06      'JUNE
TM%(CDATE%) = &H01   'FIRST
TM%(HR%) = &H03      '3 A.M.
TM%(MIN%) = &H15     '15 MINUTES
TM%(SEC%) = &H00     '0 SECONDS
GOSUB 10300          'SETALARM
    
```

2. Disable alarm interrupts:

```
TM%(HR%) = 0  
TM%(MIN%) = 0  
TM%(SEC%) = 0  
GOSUB 10300
```

```
'SETALARM
```

Purpose: Set the clock

Format: GOSUB 10500 'SETCLOCK

Remarks: Set the clock to the time and date indicated in the array TM% (previously defined).

Each component of TM% contains two BCD digits. The values passed in for MSEC% and CDSEC% in the variable TM% are ignored. The MSEC% and CDSEC% components of the clock are initialized to zero.

The YR% component is the year minus 1984.

Example:

1. Set the clock to 8:22:44 a.m. on Tuesday, July 4, 1989. (Note: the use of hex notation equates to assigning BCD values to TM%):

TM%(DAY%) = &H02	'TUESDAY
TM%(MO%) = &H07	'JULY
TM%(CDATE%) = &H04	'FOURTH
TM%(YR%) = &H05	'1989 - 1984 = 5
TM%(HR%) = &H08	'8 A.M.
TM%(MIN%) = &H22	'22 MINUTES
TM%(SEC%) = &H44	'44 SECONDS
GOSUB 10500	'SETCLOCK

Purpose: Set the periodic interrupt frequency

Format: GOSUB 10600 'SETFREQ

Remarks: Set the periodic interrupt frequency according to the value of the integer N%.

&H0002 every 100ms
 &H0004 every second
 &H0008 every minute
 &H0010 every hour
 &H0020 every day
 &H0040 every week
 &H0080 every month

When the alarm occurs, an interrupt is generated. You are responsible for setting up the interrupt vector and routine prior to calling SETALARM.

If N% is 0, the periodic interrupt is disabled (note that bit 0 is unused.)

Setting the periodic interrupt frequency causes the clock to generate an interrupt at the frequency specified.

Examples:

1. Set the periodic interrupt frequency to every minute:

```
N% = &H08
GOSUB 10600          'SETFREQ
```

2. Disable the periodic interrupt:

```
N% = 0
GOSUB 10600          'SETFREQ
```

C Language Clock Functions

The following paragraphs describe how to call the C language clock functions from a C language application program.

Table 2 - Clock Function Summary - C Language

Description	Function Syntax
Get the current alarm setting	getalarm (&get)
Get the current clock setting	getclock (&get)
Set the alarm	setalarm (&set)
Set the clock	setclock (&set)
Set the periodic interrupt frequency	setfreq (n)

The arguments &get and &set are pointers to a structure declared as follows:

```
struct time {    char msec,      /* ten thousandths of seconds */
                cdsec,     /* hundredths and tenths of seconds*/
                sec,       /* seconds */
                min,      /* minutes */
                hr,       /* hours */
                day,      /* day of the week */
                date,     /* day of the month */
                mo,       /* month */
                yr;      /* year */
};
```

The argument n is an integer variable.

If the base I/O address is not 0x02E1, the following changes must be made in clock.c:

```
struct PC2 *base = (struct PC2 *) (0x base address + 0x9406)
```

in int.asm:

```
CCR = base address + 0x9400
CDR = base address + 0x9800
```

Purpose: Get the current alarm setting

Format: `getalarm (&get)`

Remarks: Return the current alarm setting in the time structure called `get`, declared as:

```
struct time get;
```

Each component of `get` contains two BCD digits.

The day of the week is returned as a number from 1 through 7, representing Sunday through Saturday. The `yr`, `msec`, and `cdsec` components are set to zero.

Example:

1. Get the alarm setting and print it. (Note: the data is printed using hex format to convert the BCD-stored values to appropriate ASCII decimal numbers):

```
getalarm (&get);  
/* Print using hex format to get BCD values */  
printf ("Day: %x\n", get.day);  
printf ("Date: %x/%x\n", get.mo, get.date);  
printf ("Time: %x:%x:%x\n", get.hr, get.min, get.sec);
```

Purpose: Get the current clock setting

Format: `getclock (&get)`

Remarks: Return the current time, day of the week, and date in the structure called `get`, declared as:

```
struct time get;
```

Each component of `get` contains two BCD digits. The `msec`, `cdsec`, and `sec` components contain the number of seconds into the current minute to an accuracy of 0.1 millisecond: SS.MMUU.

The day of the week is returned as a number from 1 through 7, representing Sunday through Saturday. The `get.yr` component is the year minus 1984.

Example:

1. Get the clock setting and print it. (Note: the data is printed using hex format to convert the BCD-stored values to appropriate ASCII decimal numbers):

```
getclock (&get);  
/* Print using hex format to get BCD values*/  
printf ("Day: %x\n", get.day);  
printf ("Date: %x/%x/%d\n", get.mo, get.date, get.yr+84);  
printf ("Time: %x:%x:%x.%x%x\n", get.hr, get.min, get.sec,  
get.cdsec, get.msec);
```

Purpose: Set the alarm

Format: setalarm (&set)

Remarks: Set the alarm time to that indicated in the time structure called set, declared as:

```
struct time set;
```

If &set is 0, alarm interrupts are disabled.

When the alarm occurs, an interrupt is generated. You are responsible for setting up the interrupt vector and routine prior to calling SETALARM.

The values passed in for yr, msec, and cdsec in the set variable are ignored, i.e., the alarm resolution is no finer than 1 sec.

Examples:

1. Set the alarm time to 3:15:00 a.m. on June 1. (Note: the use of hex notation equates to assigning BCD values to set):

```
set.mo = 0x06;          /* June          */
set.date = 0x01;       /* first         */
set.hr = 0x03;         /* 3 a.m.       */
set.min = 0x15;       /* 15 minutes   */
set.sec = 0;           /* 0 seconds    */
setalarm (&set);
```

2. Disable alarm interrupts:

```
setalarm (0);
```

- Purpose:** Set the clock
- Format:** setclock (&set)
- Remarks:** Set the clock to the time and date indicated in the time structure called set, declared as:

```
struct time set;
```

Each component of set contains two BCD digits. The values passed in for msec and csec in the variable set are ignored. The msec and csec components of the clock are initialized to zero.

The set.yr component is the year minus 1984.

Example:

1. Set the clock to 8:22:44 a.m. on Tuesday, July 4, 1989. (Note: the use of hex notation equates to assigning BCD values to set):

```
set.day = 0x02;      /* Tuesday      */
set.mo = 0x07;      /* July         */
set.date = 0x04;    /* fourth       */
set.yr = 0x05;      /* 1989 - 1984 = 5 */
set.hr = 0x08;      /* 8 a.m.       */
set.min = 0x22;     /* 22 minutes   */
set.sec = 0x44;     /* 44 seconds   */
setclock (&set);
```

Purpose: Set the periodic interrupt frequency

Format: setfreq (n)

Remarks: Set the periodic interrupt frequency according to the value of integer n.

0x0002 every 100ms
0x0004 every second
0x0008 every minute
0x0010 every hour
0x0020 every day
0x0040 every week
0x0080 every month

When the alarm occurs, an interrupt is generated. You are responsible for setting up the interrupt vector and routine prior to calling SETALARM.

If n is 0, the periodic interrupt is disabled (note that bit 0 is unused.)

Setting the periodic interrupt frequency causes the clock to generate an interrupt at the frequency specified.

Examples:

1. Set the periodic interrupt frequency to every minute:

```
setfreq (0x8);
```

2. Disable the periodic interrupt:

```
setfreq (0);
```

8086 Assembly Language Clock Functions

The following paragraphs describe how to call the 8086 assembly language clock functions from an 8086 assembly language application program.

Table 3 - Clock Function Summary - Assembly Language

Description	Function	Parameter
Get the current alarm setting	getalarm	time
Get the current clock setting	getclock	time
Set the alarm	setalarm	time
Set the clock	setclock	time
Set the periodic interrupt frequency	setfreq	n

The structure "time", defined below, is used to set and get the clock time. The address of the first element is passed in the bx register. The constants "msec" through "yr" are used to index the structure.

; time structure:

```

time      db 0      ; ten thousandths of seconds
          db 0      ; hundredths and tenths of seconds
          db 0      ; seconds
          db 0      ; hours
          db 0      ; minutes
          db 0      ; day of the week
          db 0      ; day of the month
          db 0      ; month
          db 0      ; year
msec     equ 0
cdsec    equ 1
sec      equ 2
min      equ 3
hr       equ 4
day      equ 5
date     equ 6
mo       equ 7
yr       equ 8

```

The argument *n* is an integer variable. See the function descriptions for the specific calling sequences.

If the base I/O address is not 02E1H, the following change must be made in `clock.asm`:

`CCR = base address + 9400H`

`CDR = base address + 9800H`

Purpose: Get the current alarm setting

Format: lea bx,time
call getalarm

Remarks: Return the current alarm setting in the structure "time".

Each component of get contains two BCD digits.

The day of the week is returned as a number from 1 through 7, representing Sunday through Saturday. The yr, msec, and cdsec components are set to zero.

Example:

1. Get the alarm setting and print it:

```
lea bx,time
call getalarm
; call routine (here undefined) to print time variable
call printstruc
```

Purpose: Get the current clock setting

Format: lea bx,time
call getclock

Remarks: Return the current time, day of the week, and date in the structure "time".

Each component of get contains two BCD digits. The msec, cdsec, and sec components contain the number of seconds into the current minute to an accuracy of 0.1 millisecond: SS.MMUU.

The day of the week is returned as a number from 1 through 7, representing Sunday through Saturday. The yr component is the year minus 1984.

Example:

1. Get the clock setting and print it:

```
lea bx,time  
call getclock  
; call routine (here undefined) to print time variable  
call printstruc
```

Purpose: Set the alarm

Format: lea bx,time
call setalarm

Remarks: Set the alarm time to that indicated in the structure "time".

If 0 is passed to setalarm in place of the address of set, alarm interrupts are disabled.

When the alarm occurs, an interrupt is generated. You are responsible for setting up the interrupt vector and routine prior to calling SETALARM.

The values passed in for yr, msec, and cdsec in the time variable are ignored, i.e., the alarm resolution is no finer than 1 sec.

Examples:

1. Set the alarm time to 3:15:00 a.m. on June 1. (Note: the use of hex notation equates to assigning BCD values to time):

```
lea bx,time

mov mo[bx],06h ; June
mov date[bx],01h ; first day of month
mov hr[bx],03h ; 3 a.m.
mov min[bx],15h ; 15 minutes
mov sec[bx],0 ; 0 seconds
call setalarm
```

2. Disable alarm interrupts:

```
xor bx,bx ; zero in bx disables alarm
call setalarm
```

Purpose: Set the clock

Format: lea bx,time
call etclock

Remarks: Set the clock to the time and date indicated in the structure called "time".

Each component of time contains two BCD digits. The values passed in for msec and cdtsec in the variable time are ignored. The msec and cdtsec components of the clock are initialized to zero.

The yr component is the year minus 1984.

Example:

1. Set the clock to 8:22:44 a.m. on Tuesday, July 4, 1989. (Note: the use of hex notation equates to assigning BCD values to time):

```

lea    bx,time
mov    day[bx],02h    ; Tuesday
mov    mo[bx],07h     ; July
mov    date[bx],04h   ; fourth
mov    yr[bx],05h     ; 1989 - 1984 = 5
mov    hr[bx],08h     ; 8 a.m.
mov    min[bx],22h    ; 22 minutes
mov    sec[bx],44h    ; 44 seconds
call   setclock
    
```

Purpose: Set the periodic interrupt frequency

Format: mov dl,freq ; pass frequency in dl
 call setfreq

Remarks: Set the periodic interrupt frequency according to the value of integer n.

0002h every 100ms
0004h every second
0008h every minute
0010h every hour
0020h every day
0040h every week
0080h every month

When the alarm occurs, an interrupt is generated. You are responsible for setting up the interrupt vector and routine prior to calling SETALARM.

If n is 0 the periodic interrupt is disabled (note that bit 0 is unused.)

Setting the periodic interrupt frequency causes the clock to generate an interrupt at the frequency specified.

Examples:

1. Set the periodic interrupt frequency to every minute:

```
mov  dl,08h
call setfreq
```

320028-01

2. Disable the periodic interrupt:

```
xor    dl,dl  
call   setfreq  
call   setclock
```