NASA Technical Memorandum 87652

OPERATION OF THE HP2250 WITH THE HP9000 SERIES 200 USING PASCAL 3.0

John Perry and C. W. Stroud

FEBRUARY 1986

Publicly Released on
2/89

(NASA-TM-87652) OPERATION OF THE HP2250
WITH THE HP9000 SERIES 200 USING PASCAL 3.0
(NASA) 35 p CSCL 12A

N89-15892

Unclas
G3/59 0192930

Date for general release February 28, 1989

NASA
National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

OPERATION OF THE HP2250
WITH THE HP9000 USING
PASCAL 3.0

SUMMARY

A computer program has been written to provide an interface between the HP Series 200 desktop computers, operating under HP Standard PASCAL 3.0, and the HP2250 Data Acquisition and Control System. PASCAL 3.0 for the HP9000 desktop computer gives a number of procedures for handling bus communication at various levels. It is necessary, however, to reach the lowest possible level in PASCAL to handle the bus protocols required by the HP2250. This makes programming extremely complex since these protocols are not documented. The program described herein solves those problems and allows the user to immedately program, simply and efficiently, any measurement and control language (MCL/50) application with a few procedure calls. The complete set of procedures is available on a 5 1/4" diskette from Cosmic. Included in this group of procedures is an Exerciser which allows the user to exercise his HP2250 interactively. The Exerciser operates in a fashion similar to the Series 200 operating system programs, but is adapted to the requirements of the HP2250.

The requirements for linking to a user's programs are described in detail when the diskette is used as received. The procedure for communicating with the HP2250 is very straightforward, once a user's program has been debugged and compiled. The programs on the diskette and the user's manual assume the user is acquainted with both the MCL/50 programming language and HP Standard PASCAL 3.0 for the HP series 200 desktop computers.

I. Introduction

The Hewlett Packard HP2250 is a high performance data acquisition and control system that permits users to control and monitor large process control systems using a host computer to interface with the HP2250. The HP2250's processor is programmed in a dedicated programming language, MCL/50 (ref. 1). Communication is via the IEEE 488 standard General Purpose Interface Bus, and 14 secondary addresses give the host access to all data defined by user programs downloaded to the HP2250 and to numerous status blocks defined within the system. The HP2250 can be operated by the HP9836A computer from BASIC. However, this BASIC is too slow for many applications, and no PASCAL-based operating program is available.

A program named PASCAL_HP2250 has been written to give a user full access to the features of the HP2250 data acquisition system by using a few simple, efficient procedure calls from PASCAL 3.0 (ref. 2). This paper gives a functional explaination of PASCAL_HP2250, and explains how to operate the system. The paper is intended to serve as a user's reference and operating manual for the program, which can be obtained from COSMIC (ref. 3).

The main bus address, secondary 0 (zero), allows loading and execution of tasks written in MCL. These tasks may be temporary tasks executed immediately or they may be permanent "resident" tasks identified by a TASK command.

Secondary addresses 1, 2, 3, and 4 give the host computer access to various MCL status blocks. The status may only be read: the host may not modify any status variables. The host must, however, write to Secondary 3 to tell it which task status is desired.

Secondaries 5 and 6 write to and read from MCL buffers in real time, i.e., they immediately receive the current values, and these values may change at any time.

Secondaries 7 and 8 write to and read from MCL variables in real time.

Secondary address 9 allows the host to write down-loaded precompiled machine code (for the HP1000 computer) subroutines to the HP2250.

Secondary address 10 is not used by the HP2250.

Secondary addresses 11, 12, 13, and 14 are "Ports". MCL tasks may assign a buffer to a port, whereupon that buffer may not be modified by MCL I/O commands until it is read by the host computer. Buffers released to ports are thus protected from intervening modification and are consequently not real time.

Pascal 3.0 for the HP9000 series 200 desk top computer gives a number of procedures for handling bus communications at various levels. It is necessary, however, to reach the lowest posssible level in Pascal in order to handle the bus protocols required by the HP2250. This makes programming extremely complex--particularly since these protocols are not documented.

Pascal_HP2250 raises the programming level, giving the user full access to all features of MCL tasks and permitting the user to program simply and efficiently any MCL application with a few simple, efficient procedure calls (figure 1). It does not handle secondary address 9; however, facilities are present in the module which allow a user to build a procedure analogous to TRANSFER_TASK which will do this. Such a user must have ready access to an HP1000 computer, and have a real need for machine language routines.

This report assumes that the user is acquainted with both MCL/50, the HP2250's programming language, and HP Standard Pascal 3.0 for the HP Series 200 desktop computers. The complete set of procedures is available on a 5 1/4" diskette from COSMIC. Information included as appendix A is a set of instructions for linking the program as received from COSMIC to a user program in Pascal. Appendix B is a complete listing of the program with a minimal driving program and a sample HP2250 program for demonstration of the Transfer_Task procedure.

II. Decription of Program:

A. Data Structures

The data structures in PASCAL_HP2250 are determined by the HP2250's data structures.

Type com_strg is an 80-character command string; its primary use is to give a single short command. More complex main tasks may be entered, however, by simply putting commands into the main address line by line without entering the "!" character.

Status_type and Interr_type are determined by the structure of their secondary addresses' data; their use is straightforward.

Buffer_type is a record having a count field "count" and a data array field "data". The count field gives the number of valid items in the data array field.

B. Procedures

1. Exported Procedures.

Each of the exported procedures performs a specific function, and most completely handle one secondary address. These procedures are:

Init_2250: sets up the internal bus addresses for the module's other procedures. Must be executed before any other procedure.

Read_main: reads data from the main address.

Write_main: writes an MCL command string to the main address.

Transfer_task: reads an MCL task from a disk file and writes it to the main address.

System_status: reads system status from secondary address 1.

Main_status: reads main task status from secondary address 2.

Resident_status: reads resident task status from secondary address 3.

Interrupt_status: reads interrupt status from secondary address 4.

Write_buff: writes data to an MCL buffer via secondary address 5.

Read_buff: reads data from an MCL buffer via secondary address 6.

Write_variable: writes data to a sequence of MCL variables via secondary address 7. Since a number of consecutive variables may be written in one command, buf_type is used to hold the data to be written.

Read_variable: reads data from a sequence of MCL variables via secondary address 8. Uses buf_type in the same way Write_Variable does.

Read_port: reads data from one of the four port addresses.

2. Internal Procedures

The internal procedures are not available outside the module Pascal_hp2250. They are very useful within it, however, for making the procedures and the bus protocols much easier to understand and use.

Talk_to_sec: sets up the bus to allow the host computer to talk to the HP2250 secondary addresses.

Listen_to_sec: sets up the bus to allow the host computer to listen.

Word_eoi: divides output 16-bit integers into two bytes so that the EOI control line drops with the last byte.

3. Exerciser

The exerciser is a procedure which allows the user to handle interactively all the facilities in module Pascal_HP2250. It prompts the user for a single-character command in the fashion of the operating system, and depending upon that command executes or prompts for further input. The exerciser in combination with user-applied MCL commands can exercise every facet of HP2250 operation under Pascal 3.0 except machine language subroutine downloading.

The only external code needed to operate the exerciser is a program calling Init_2250 with the correct interface select code and bus address for the user's system, and a call to the exerciser. The user may terminate the program in any convenient manner outside the exerciser.

The main program HPROG supplied on the diskette illustrates the use of Exerciser and Init_2250 on the system described herein. If the user's addresses are different, it is necessary to change only the declarations.

III. Use of Program

A. Data Structures

1. Constants: Maxram=16384 is the maximum user ram available in the HP2250 system processor. It is thus impossible to have a larger data array than Maxram.

   Port_a--Port_d: The secondary addresses of the respective ports.

2. Types: Com_Strg=String[80]; String variable type that handles command strings to the hp2250 main address and the file name for task transfers. It holds an alphanumeric string up to 80 characters long which the user's program must fill.

```
buffer_type=record count: 0..maxram;
                   data:  array[1..maxram] of integer;
            end;
```

Buffer_type is a general-purpose integer array with associated count variable. All secondary addresses return integers, and some require integer inputs. Since it is not possible to predict the exact size each user program will require, buffer_type is defined to include the largest possible data array. The user must define his own record type similar to buffer_type, but with an appropriate data array size.

The count field in the user's record must always reflect the true data count in the array; it is strongly recommended that both the count field and the data array size be limited with an appropriate constant size in the same manner as buffer_type's were. Since the compiler does not generate value checking for the "anyvar" declarations in buffer_type usage, careless use of this type can crash your system. It is further recommended that the count field be updated immediately whenever the data field is changed: this allows the user to keep track of the true size of the data array.

The procedures that use buffer_type as an output take care of these operations internally; the procedures that use buffer_type as input depend upon the count field for their operation. This affords some protection from trouble, as long as the user is careful to assign values to the data field only in concert with correct updating of the count field.

Status_type=array [1..8] of integer;
All status addresses except the interrupt status have an eight-element integer array as output. These are read-only arrays which are used as output by their respective procedures.

Interr_type=array [1..16] of integer;
A 16-element array used as output by the interrupt_status procedure.

Port type=port a..port d;
Port type defines the addresses of the ports corresponding to their namesakes. It is used as input by the Read_Port procedure.

B. Procedures

Init_2250 (select_code: type_isc; addr: type_hpib_addr;);

Init_2250 sets up the addressing structure for the user's system. It must be the first call to the module Pascal_HP2250.

Select_code will be 7 if the internal hpib is used; it will usually be 8 if a single external hpib card is used. Addr may be determined by the user for his system.

Read_main (anyvar data: buffer_type);

Read_main reads the main result buffer from the HP2250's primary address. It leaves the data in the lowest elements of data.data, and leaves the number of elements in data.count.

Write_main (strg: com strg);

Write_main sends a user-specified MCL command to the primary address of the HP2250. Refer to the HP2250 Programmer's manual (ref. 1) for valid MCL commands.

Transfer_task (taskfile: com_strg);

Transfer_Task allows the user to hold complete tasks on disk and transfer them by specifying the file name holding the task. Taskfile must be the valid file specifier of an existing text file; this file must contain an MCL task.

```
System_Status (var status:  status_type);
Main_Status (var status:  Status_type);
```

System_Status and Main_Status get the status arrays from secondaries 1 and 2, respectively. Their parameters are output only, and the status may be read at any time by calling the procedure, then examining the array.

```
Resident_Status (task:integer; var status: status_type);
```

Resident_Status differs from Main_status only in that it may retrieve the status of any task--not just the main task. It therefore requires an input parameter, task, to tell it which status to read; status may then be read from secondary 3 like main_status.

```
Interrupt_Status (var interrupts:  interr_type);
```

Interrupt_Status reads the 16-element interrupt status array from secondary address 4. Interrupts is an output array only.

```
Write_buff (bufno:integer: anyvar data:  buffer_type);
```

Write_buff writes via secondary 5 the number of elements given in data.count from the array data.data to the MCL buffer number given in bufno.

Bufno must contain the integer name of the desired MCL buffer; data.count must contain the number of words to be written; and data.data must contain the data values to be written.

```
Read_buff (bufno:integer; anyvar data:  buffer_type);
```

Read_buff reads into variable data.data via secondary 6 data from the MCL buffer specified in bufno.

Bufno must contain the integer name of the desired MCL buffer; data.count must contain the number of words to be read; and data.data will contain upon return to the caller the data values in the buffer.

```
Write_variable (varno:integer; anyvar data: buffer_type);
```

Write_variable writes via secondary 7 the number of elements in data.count from the array data.data to a sequence of MCL variables starting with that named in "varno" and continuing until the array is exhausted.

Varno must contain the integer name of the first desired MCL variable; data.count must contain the number of data to be transferred, therefore the number of variables to be written to; and data.data must contain the values to be written. There must be exactly as many values in data.data as are specified by data.count.

Read_variable (varno:integer; anyvar data: buffer_type);

Read_variable reads, via secondary 8, the number of elements in data.count from a sequence of variables starting with that named in varno and continuing for the number of variables given in data.count. The data are read into the array data.data.

Varno must contain the integer name of the first desired MCL variable; data.count must contain the number of variables to be read; and data.data will contain the values, in order, upon return from Read_Variable.

Read_port (port:port_type; anyvar data: buffer_type);

Read_port reads a buffer from the port named in "port" into "data". Before any data can be expected at the port, an MCL task running in the HP2250 must have executed a RELEASE command to the port named. The array data.data must be declared large enough to hold the entire expected buffer.

Port must contain an element of the integer subrange port_a..port_d (11..14); upon return, data.count will contain the number of data elements found at the port; data.data will contain the data found.

Exerciser;

The Exerciser allows the user to exercise his HP2250 interactively. It operates in a fashion similar to the operating system programs; however, it is augmented to reflect the different nature of the HP2250.

Several levels of prompts must be traversed before a command is executed; the number of levels and the details of the level structure depend upon which selection is made at each level. This is not complex, however, because the levels and the prompts are arranged in an orderly, logical fashion.

Each level's prompt gives a set of single-character commands possible within its level; upon receipt of a valid character, the level prints out the remainder of the command's text and goes to the next level.

Thus:

r)ead, w)rite, t)ask, s)tatus, q)uit: expects a single character, "r", "w", "t", "s", or "q".

If the operator types "r", he will see upon the screen

```
r)ead, w)rite, t)ask, s)tatus q)uit:  Read
m)ain, v)ariable, b)uffer, p)ort:
```

If he then types "m", he will see

```
r)ead, w)rite, t)ask, s)tatus, q)uit:  read
m)ain, v)ariable, b)uffer, p)ort:  main
          0
```

Since read_main requires no more input, the exerciser immediately reads the main result data, which in this case was a single word zero. Characters will be accepted until a "q" or "Q" is typed; the exerciser will then return to the calling program.

Some levels require input data from the keyboard. In all cases the prompts are straightforward and depend upon the parameters required by the procedures they exercise.

CONCLUDING REMARKS

This program solves the problem of communication between the HP9000 series 200 computer and the HP220 data acquisition and control system. The program removes the need for low level access to the Pascal input/output system, making all facilities of the HP2250 available to the Pascal user in simple procedure calls. The only facility not currently supported is machine-language down loading. The complete set of procedures is available from COSMIC on a 5 1/4" diskette. Included in this group of procedures is an Exerciser, which allows the user to exercise his HP2250 interactively.

APPENDIX A

Linking to a User Program

If the diskette is used as it is, the requirements for linking are given in the Pascal 3.0 User's Manual for HP Series 200 Computers. In summary, the requirements for this program are:

In the text of your Pascal source, insert the line

```
$ search 'PA2250:PASC_2250'$.
```

This tells the compiler where to find the module. In the declaration part of your source, insert

```
Import Pascal_HP2250;
```

All the export declaration will then be available for linking to your program.

After the user program has been debugged and compiled, it will be necessary to link the modules using the Librarian.

An example is given on the diskette in the stream file

```
PA2250:HPROG.LNK.TEXT
```

for the sample program HPROG on the diskette. HP.CODE is the linked, executable version given for this document.

APPENDIX B

The program HP (file HPROG) is a minimum driving program for module Pascal_HP2250. If the HPIB select code or device number are different, the sel_code and address must be changed to match your system's. The program will then execute properly.

This program can, in fact, be used to check out your system's operation, and to train personnel in the use of the HP2250, since its only limitation is its lack of a procedure to transfer machine code to the HP2250. Every other facility is available to the user, depending upon the input/output cards installed in the system.

THE FOLLOWING

PAGES ARE

PROGRAM LISTINGS

```
Pascal [Rev 3.0  6/ 4/84] HPROG.TEXT                 4-Dec-85 14:35:58 Page 1

     1:D          0 $list on$
     2:D          0 $lines 58$
     3:D          0 $pagewidth 80$
     4:D          0 $search 'PA2250:PASC_2250'$
     5:D          0 PROGRAM HP (input, output);
     6:S
     7:D          1 import  Pascal_HP2250, iocomasm, iodeclarations;
     8:S
     9:D          1 const   sei_code=8;      {for the author's system}
    10:D          1         address=5;
    11:S
    12:D     -1   1 var     ch: char;
    13:S
    14:C          1 begin   {hp}
    15:C          1   init_2250 (sei_code, address);
    16:C          1   exerciser;
    17:C          1 end.    {hp}

No errors. No warnings.
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
     1:D        0 $list on$
     2:D        0 $sysprog$
     3:D        0 $lines 58$
     4:D        0 $pagewidth 80$
     5:S
     6:S          MODULE  Pascal_HP2250:          {interface between 9836 Pascal
2.1
     7:D        1                                                   and hp2250}
     8:S
     9:S
    10:D        1 Import  hpib_0, hpib_1, hpib_2, hpib_3, general_0, general_1,
general_2,
    11:D        1         general_3, general_4, iocomasm, iodeclarations, fs;
    12:S
    13:D        1 Export
    14:D        1         const   maxram=16384;   {maximum RAM available to user
}
    15:D        1                 port_a=11;      {secondary addresses of}
    16:D        1                 port_b=12;      {  ports in HP2250     }
    17:D        1                 port_c=13;
    18:D        1                 port_d=14;
    19:S
    20:S
    21:D        1         type    com_strg=string[80];    {command string for al
fa use}
    22:D        1
    23:S                            {User of buffer_type must be careful in call
ing program
    24:D        1                       that count ALWAYS reflects true data cou
nt!}
    25:D        1                 buffer_type=record
    26:D        1                             count: 0..maxram;       {data ite
ms in array}
    27:D        1                             data: array [1..maxram] of integ
er; {data array}
    28:D        1                         end;        {buffer_type}
    29:S
    30:D        1                 status_type=array [1..8] of integer;    {statu
s data array}
    31:D        1                 interr_type=array [1..16] of integer;   {inter
rupt data}
    32:D        1                 port_type=port_a..port_d;       {secondary add
ress of ports}
    33:S
    34:S
    35:D        1 $page$
    36:D        1 $list on$
    37:D        1 procedure      init_2250 (select_code: type_isc; addr: type_h
pib_addr);
    38:D        1 procedure      read_main (anyvar data: buffer_type);
    39:D        1 procedure      write_main (strg: com_strg);
    40:D        1 procedure      transfer_task (taskfile: com_strg);
    41:D        1 procedure      system_status (var status: status_type);
    42:D        1 procedure      main_status (var status: status_type);
    43:D        1 procedure      resident_status (task: integer;
    44:D        2                                           var status: status_typ
e);
    45:D        1 procedure      interrupt_status (var interrupts: interr_type)
;
    46:D        1 procedure      write_buff (bufno: integer; anyvar data: buffe
r_type);
    47:D        1 procedure      read_buff (bufno: integer; anyvar data: buffer
_type);
    48:D        1 procedure    write_variable (varno: integer; anyvar data: buff
er_type);
    49:D        1 procedure     read_variable (varno: integer; anyvar data: buff
er_type);
    50:D        1 procedure     read_port (port: port_type; anyvar data: buffer_
type);
    51:D        1 procedure      exerciser;      {exercise module interactively
}
    52:S
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
    53:D        1 $page$
    54:D        1 $list on$
    55:D        1 Implement
    56:S
    57:D    -2  1 var     card: type_isc;         {hpib card interface select co
de}
    58:D    -6  1         bus_addr, my_addr: type_hpib_addr;
    59:D    -8  1         dev:type_device;        {composite address for certain
 commands}
    60:S
    61:S
    62:D    -8  1 {set up hpib to talk to a secondary or the main address}
    63:D        1 procedure       talk_to_sec (sec: type_hpib_addr);
    64:S
    65:C        2 begin
    66:C        2   unlisten (card); talk (card, my_addr);
    67:C        2   listen (card, bus_addr); if sec<>0 then secondary (card, sec
);
    68:C        2 end;    {talk_to_sec}
    69:S
    70:S
    71:D    -8  1 {set up hpib to listen to a secondary or the main address}
    72:D        1 procedure       listen_to_sec (sec: type_hpib_addr);
    73:S
    74:C        2 begin
    75:C        2   unlisten (card); talk (card, bus_addr);
    76:C        2   if sec<>0 then secondary (card, sec);
    77:C        2   listen (card, my_addr); if sec<>0 then secondary (card, sec)
;
    78:C        2 end;    {listen_to_sec}
    79:S
    80:S
    81:D    -8  1 {put eoi signal onto last byte of a binary message}
    82:D        1 procedure       word_eoi (word: integer);
    83:S
    84:D        2 const   byte_size=256;
    85:S
    86:C        2 begin
    87:C        2   writechar (card, chr (word div byte_size));
    88:C        2   set_hpib (card, eoi_line);
    89:C        2   writechar (card, chr (word mod byte_size));
    90:C        2 end;    {word_eoi}
    91:S
    92:S
    93:D    -6  1 {set up select code, bus address, and device address for 2250}

    94:D        1 procedure       init_2250 (select_code: type_isc; addr: type_h
pib_addr);
    95:S
    96:C        2 begin
    97:C        2   card:=select_code; bus_addr:=addr; dev:=card*100+bus_addr;
    98:C        2   my_addr:=my_address (card);
    99:C        2 end;    {init_2250}
   100:S
   101:S
   102:D    -8   1 $page$
   103:D    -8   1 $list on$
   104:D    -8   1 {read main address message from hp2250}
   105:D         1 procedure        read_main (anyvar data: buffer_type);
   106:S
   107:D    -8   2 var      i, words: integer;
   108:D   -40   2          status: status_type;
   109:S
   110:C         2 begin
   111:C         2   system_status (status);
   112:C         2   listen_to_sec (0);
   113:C         2   for i:=1 to status[4] do readword (card, data.data[i]);
   114:C         2   data.count:=status[4];
   115:C         2   untalk (card); unlisten (card);
   116:C         2 end;     {read_main}
   117:S
   118:S
   119:D    -8   1 {write a command to the main address}
   120:D   -82   1 procedure        write_main (strg: com_strg);
   121:S
   122:D   -90   2 var      i, words: integer;
   123:S
   124:C         2 begin
   125:C         2   writestring (dev,strg); set_hpib (card, eoi_line);
   126:C         2   writechar (card, chr(10));
   127:C         2   untalk (card); unlisten (card);
   128:C         2 end;     {write_main}
   129:S
   130:S
   131:D    -8   1 {write task to main address from a disk file}
   132:D   -82   1 procedure        transfer_task (taskfile: com_strg);
   133:S
   134:D  -746   2 var      task: text;     {file containing task code--must be ty
pe text}
   135:D  -828   2          command: com_strg;      {name of file--form XXXXXXXX.T
EXT}
   136:S
   137:C         2 begin
   138:C         2   reset (task, taskfile);
   139:C         2   while not eof(task) do
   140:C         3   begin
   141:C         3     readln (task, command); writestringln (dev, command);
   142:C         3   end;  {while}
   143:C         2   set_hpib (card, eoi_line); writechar (card, chr (10));
   144:C         2   untalk (card); unlisten (card);
   145:C         2 end;     {transfer_task}
   146:S
   147:S
```

```
    148:D     -8   1 $page$
    149:D     -8   1 $list on$
    150:D     -8   1 {get system status from secondary 1}
    151:D          1 procedure       system_status (var status: status_type);
    152:S
    153:D     -4   2 var i: integer;
    154:S
    155:C          2 begin
    156:C          2   listen_to_sec (1);
    157:C          2   for i:=1 to 8 do readword (card, status[i]);
    158:C          2   untalk (card); unlisten (card);
    159:C          2 end;    {system_status}
    160:S
    161:S
    162:D     -8   1 {get main task status from secondary 2}
    163:D          1 procedure       main_status (var status: status_type);
    164:S
    165:D     -4   2 var i: integer;
    166:S
    167:C          2 begin
    168:C          2   listen_to_sec (2);
    169:C          2   for i:=1 to 8 do readword (card, status[i]);
    170:C          2   untalk (card); unlisten (card);
    171:C          2 end;    {main_status}
    172:S
    173:S
    174:D     -8   1 {get resident task status from secondary 3}
    175:D          1 procedure       resident_status (task: integer;
    176:D          2                                           var status: status_typ
e);
    177:S
    178:D     -4   2 var i: integer;
    179:S
    180:C          2 begin
    181:C          2   talk_to_sec (3);
    182:C          2   word_eoi (task);
    183:C          2   untalk (card); unlisten (card);
    184:C          2   listen_to_sec (3);
    185:C          2   for i:=1 to 8 do readword (card, status[i]);
    186:C          2   untalk (card); unlisten (card);
    187:C          2 end;    {resident_status}
    188:S
    189:S
```

```
   190:D    -8  1 $page$
   191:D    -8  1 $list on$
   192:D    -8  1 {get interrupt status from secondary 4}
   193:D        1 procedure       interrupt_status (var interrupts: interr_type)
;
   194:S
   195:D    -4  2 var i: integer;
   196:S
   197:C        2 begin
   198:C        2   listen_to_sec (4);
   199:C        2   for i:=1 to 16 do readword (card, interrupts[i]);
   200:C        2   untalk (card); unlisten (card);
   201:C        2 end;    {interrupt_status}
   202:S
   203:S
   204:D    -8  1 {write data to buffer with secondary 5}
   205:D        1 procedure       write_buff (bufno: integer; anyvar data: buffe
r_type);
   206:S
   207:D    -4  2 var i: integer;
   208:S
   209:C        2 begin
   210:C        2   talk_to_sec (5);
   211:C        2   writeword (card, bufno);
   212:C        2   writeword (card, data.count);
   213:C        2   for i:=1 to data.count do writeword (card, data.data[i]);
   214:C        2   untalk (card); unlisten (card);
   215:C        2 end;    {write_buff}
   216:S
   217:S
   218:D    -8  1 {read data from buffer with secondary 6}
   219:D        1 procedure       read_buff (bufno: integer; anyvar data: buffer
_type);
   220:S
   221:D    -4  2 var i: integer;
   222:S
   223:C        2 begin
   224:C        2   talk_to_sec (6);
   225:C        2   writeword (card, bufno);
   226:C        2   word_eoi (data.count);
   227:C        2   untalk (card); unlisten (card);
   228:C        2   listen_to_sec (6);
   229:C        2   for i:=1 to data.count do readword (card, data.data[i]);
   230:C        2   untalk (card); unlisten (card);
   231:C        2 end;    {read_buff}
   232:S
   233:S
   234:D     -8   1 $page$
   235:D     -8   1 $list on$
   236:D     -8   1 {write data to variable with secondary 7}
   237:D          1 procedure    write_variable (varno: integer; anyvar data: buff
er_type);
   238:S
   239:D     -4   2 var      i: integer;
   240:S
   241:C          2 begin
   242:C          2   talk_to_sec (7);
   243:C          2   writeword (card, varno);
   244:C          2   writeword (card, data.count);
   245:C          2   for i:=1 to data.count-1 do writeword (card, data.data[i]);
   246:C          2   word_eoi (data.data[data.count]);
   247:C          2   untalk (card); unlisten (card);
   248:C          2 end;    {write_variable}
   249:S
   250:S
   251:D     -8   1 {read data from variable with secondary 8}
   252:D          1 procedure     read_variable (varno: integer; anyvar data: buff
er_type);
   253:S
   254:D     -4   2 var      i: integer;
   255:S
   256:C          2 begin
   257:C          2   talk_to_sec (8);
   258:C          2   writeword (card, varno);
   259:C          2   word_eoi (data.count);
   260:C          2   untalk (card); unlisten (card);
   261:C          2   listen_to_sec (8);
   262:C          2   for i:=1 to data.count do readword (card, data.data[i]);
   263:C          2   untalk (card); unlisten (card);
   264:C          2 end;    {read_variable}
   265:S
   266:S
   267:S
   268:D     -8   1 {read data from port with secondary 11-14}
   269:D          1 procedure     read_port (port: port_type; anyvar data: buffer_
type);
   270:S
   271:D     -4   2 var      i: integer;
   272:D    -36   2          status: status_type;
   273:S
   274:C          2 begin
   275:C          2   system_status (status);
   276:C          2   data.count:=status[port-6];
   277:C          2   listen_to_sec (port);
   278:C          2   for i:=1 to data.count do readword (card, data.data[i]);
   279:C          2   untalk (card); unlisten (card);
   280:C          2 end;    {read_port}
   281:S
   282:S
   283:S
   284:D     -8   1 $page$
   285:D     -8   1 $list on$
   286:D          1 procedure       exerciser;       {exercise module interactively
}
   287:S
   288:D          2 const   cr=chr(13);      {carriage return char}
   289:S
   290:D    -32   2 var     status: status_type;
   291:D    -60   2         row, col, i, j, varno, bufno, task: integer;
   292:D    -61   2         ch: char;
   293:D   -224   2         command, filename: com_strg;
   294:D   -224   2         data: record count: 0..32;
   295:D   -224   2                      data: array [1..32] of integer;
   296:D   -354   2               end;      {data record}
   297:D   -418   2         interr: interr_type;
   298:D   -420   2         port: port_type;
   299:S
   300:C          2 begin
   301:C          2   repeat
   302:C          3   for i:=0 to 10 do begin
   303:C          4   fgotoxy (output, 0,i); for j:=1 to 11 do write ('          ');
   304:C          4   end;
   305:C          3   fgotoxy (output, 0,1); write ('r)ead, w)rite, t)ask, s)tatus
, q)uit: ');
   306:C          3   fgetxy (output, col, row);
   307:C          3   repeat
   308:C          4     fgotoxy (output, col, row); read (ch)
   309:C          4   until ch in ['r','R','w','W','t','T','s','S','q','Q'];
   310:S
   311:C         3 $page$
   312:C         3 $list on$
   313:C         3   case ord (ch) of
   314:S
   315:C         4     {read...}
   316:C         4     ord('r'), ord ('R'): begin
   317:C         4       writeln ('ead '); write ('  m)ain, v)ariable, b)uffer, p)
ort: ');
   318:C         4       fgetxy (output, col, row);
   319:C         4       repeat
   320:C         5         fgotoxy (output, col, row); read (ch);
   321:C         5       until ch in ['m','M','v','V','b','B', 'p', 'P'];
   322:C         4       case ord (ch) of
   323:S
   324:C         5         {read main}
   325:C         5         ord ('m'), ord ('M'):
   326:C         5         begin
   327:C         5           writeln ('ain '); read_main (data);
   328:C         5           for i:=1 to data.count do write (data.data[i]:8); wr
iteln
   329:C         5         end;    {ord ('m'), ord ('M')}
   330:S
   331:C          5 $page$
   332:C          5 $list on$
   333:C          5          {read variable}
   334:C          5          ord ('v'),ord ('V'): begin
   335:C          5            writeln ('ariable ');
   336:C          5            write ('start variable, # variables: '); readln (var
no.data.count);
   337:C          5
   338:C          5            read_variable (varno, data);
   339:C          5            for i:=1 to data.count do write (data.data[i]:8);
   340:S          5          end;    {ord ('v'), ord ('V')}
   341:C
   342:C          5          {read buffer}
   343:C          5          ord ('b'),ord ('B'): begin
   344:C          5            writeln ('uffer ');
ta.count);        5            write ('buffer number, # words: '); readln (bufno,da
   345:C          5
   346:C          5            read_buff (bufno, data);
   347:C          5            for i:=1 to data.count do write (data.data[i]:8);
   348:S          5          end;    {ord ('b'), ord ('B')}
   349:C
   350:C          5          {read port}
   351:C          5          ord ('p'),ord ('P'): begin
   352:C          5            writeln ('ort ');
   353:C          5            write ('port name (A,B,C,D): ');
   354:C          5            fgetxy (output, col, row);
   355:C          6            repeat
   356:C          6              fgotoxy (output, col, row); read (ch);
   357:C          5            until ch in ['A','B','C','D'];
   358:C          5            port:=ord(ch)-54;
   359:C          5            read_port (port, data);
   360:C          5            for i:=1 to data.count do write (data.data[i]:8);
   361:C          5          end;    {ord ('p'), ord ('P')}
   362:C          5
   363:C          5          otherwise {do nothing}
   364:C          5        end;      {case ord (ch)}
   365:S          4      end;        {ord ('r'), ord ('R')}
   366:C          4 $page$
   367:C          4 $list on$
   368:C          4      {write...}
   369:C          4      ord('w'), ord ('W'): begin
   370:C          4        writeln ('rite');
   371:C          4        write (cr, ' m)ain, v)ariable, b)uffer: ');
   372:C          4        fgetxy (output, col, row);
   373:C          4        repeat
   374:C          5          fgotoxy (output, col, row); read (ch)
   375:C          5        until ch in ['m','M','v','V','b','B'];
   376:C          4        case ord (ch) of
   377:S
   378:C          5          {write main}
   379:C          5          ord ('m').ord ('M'): begin
   380:C          5            writeln ('ain');
   381:C          5            write ('MCL command: '); readln (command); write_mai
n (command)
   382:C          5          end;    {ord ('m'), ord ('M')}
   383:S
   384:C          5          {write variable}
   385:C          5          ord ('v'),ord ('V'): begin
   386:C          5            writeln ('ariable');
   387:C          5            write ('start variable, # variables: '); readln (va
rno, data.count);
   388:C          5            write (data.count:2, ' values: ');
   389:C          5            for i:=1 to data.count-1 do read (data.data[i]);
   390:C          5            readln (data.data[data.count]);
   391:C          5            write_variable (varno, data)
   392:C          5          end;    {ord ('v'), ord ('V')}
   393:S
   394:C          5          {write buffer}
   395:C          5          ord ('b').ord ('B'): begin
   396:C          5            writeln ('uffer');
   397:C          5            write ('buffer number, # words: '); readln (bufno,da
ta.count);
   398:C          5            write (data.count:2, ' values: ');
   399:C          5            for i:=1 to data.count-1 do read (data.data[i]);
   400:C          5            readln (data.data[data.count]);
   401:C          5            write_buff (bufno. data)
   402:C          5          end;    {ord ('b'), ord ('B')}
   403:C          5
   404:C          5          otherwise {do nothing}
   405:C          5        end;      {case ord (ch)}
   406:C          4      end;        {ord('w'), ord ('W')}
   407:S
   408:C          4      {transfer task...}
   409:C          4      ord('t'), ord ('T'): begin
   410:C          4        writeln ('ransfer task');
   411:C          4        write ('task filename: ');readln (filename); transfer_ta
sk (filename);
   412:C          4        writeln (filename, ' sent');
   413:C          4      end;        {ord('t'), ord ('T')}
   414:S
   415:C        4 $page$
   416:C        4 $list on$
   417:C        4     {status of...}
   418:C        4     ord('s'), ord ('S'): begin
   419:C        4       writeln ('tatus of:');
   420:C        4       write (' s)ystem, m)ain, r)esident, i)nterrupt: ');
   421:C        4       fgetxy (output, col, row);
   422:C        4       repeat
   423:C        5         fgotoxy (output, col, row); read (ch);
   424:C        5       until ch in ['s','S','m','M','r','R', 'i', 'I'];
   425:C        4       case ord (ch) of
   426:S
   427:C        5         {status of system}
   428:C        5         ord ('s'), ord ('S'):
   429:C        5         begin
   430:C        5           writeln ('ystem'); system_status (status);
   431:C        5           for i:=1 to 8 do write (status[i]:4); writeln
   432:C        5         end;    {ord ('s'), ord ('S')}
   433:S
   434:C        5         {status of main}
   435:C        5         ord ('m'),ord ('M'): begin
   436:C        5           writeln ('ain'); main_status (status);
   437:C        5           for i:=1 to 8 do write (status[i]:4); writeln
   438:C        5         end;    {ord ('m'), ord ('M')}
   439:S
   440:C        5         {status of resident task}
   441:C        5         ord ('r'),ord ('R'): begin
   442:C        5           writeln ('esident task');
   443:C        5           write ('task: '); readln (task); resident_status (ta
sk, status);
   444:C        5           for i:=1 to 8 do write (status[i]:4); writeln
   445:C        5         end;    {ord ('r'), ord ('R')}
   446:S
   447:C        5         {status of interrupts}
   448:C        5         ord ('i'),ord ('I'): begin
   449:C        5           writeln ('nterrupt');
   450:C        5           interrupt_status (interr);
   451:C        5           for i:=1 to 16 do write (interr[i]:4); writeln
   452:C        5         end;    {ord ('i'), ord ('I')}
   453:C        5
   454:C        5         otherwise {do nothing}
   455:C        5       end;        {case ord (ch)}
   456:C        4     end;          {ord ('s'), ord ('S')}
   457:S
   458:C        4     otherwise {do nothing}
   459:C        4   end;  {case ord (ch)}
   460:C        3   until ch in ['q','Q']
   461:S
   462:C        3 end;    {exerciser}
   463:S
   464:C        1 end.    {Pascal_HP2250}

No errors. No warnings.
                    ***** Nonstandard language features enabled *****
```

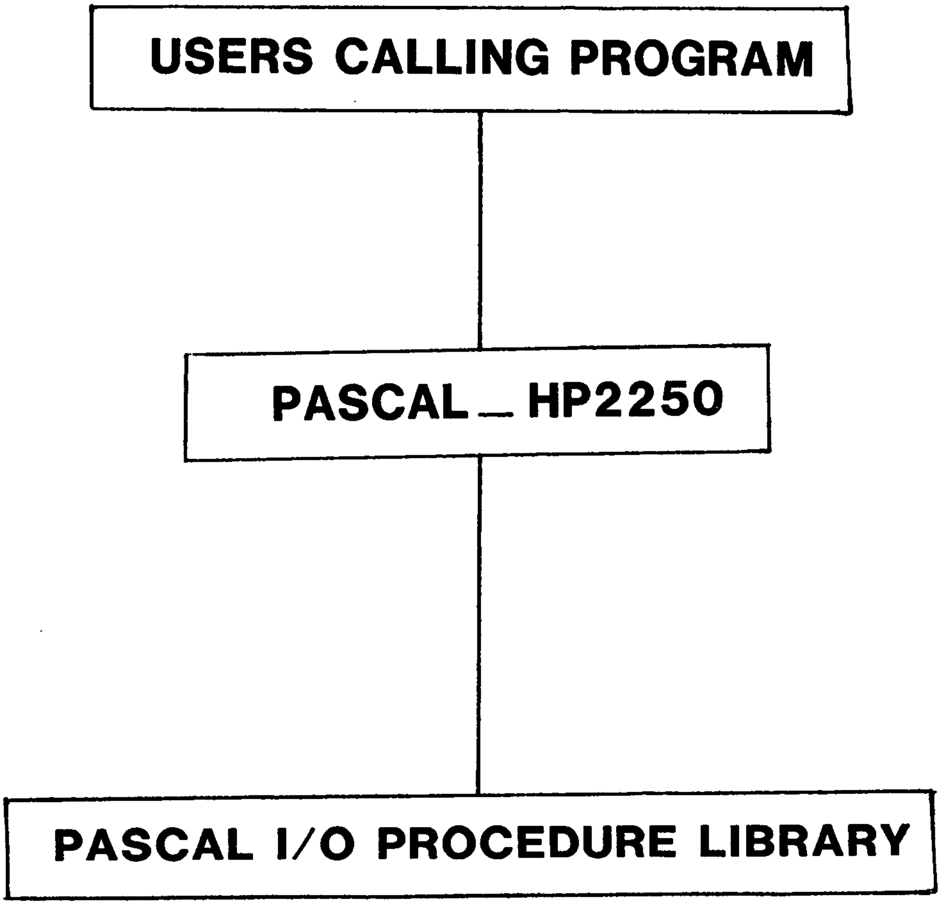
```
**STREAMFILE 'HPROG.LNK.TEXT'**
  from operating system level use stream command on this file
    to link HPROG and PASC_2250.



LOHP
IHPROG
AIPASC_2250
ALKQ



**FILE INITDAS.TEXT**
  use this as source for Transfer_task command. Defines 10 tasks,
    10 variables, 10 buffers each 10 words long, and asks for
      amount of available memory.


ntasks (0), ntasks (10); dimension (20,10,32,12,10)
clb(1); clb (2); clb (3); aon (1)!
```

REFERENCES

1. Hewlett-Packard Measurement and Control Processor Programmer's Manual, HP part number 25580-90001, Mar. 1981.

2. Hewlett-Packard Pascal Language Reference for the HP9000 series 200 computers. HP part number 98615-90050, Feb. 1984.

3. Computer Software Management and Information Center, 112 Barrow Hall, The University of Georgia, Athens, GA, 30602. VPD 7744/7-82.



[redacted] Block diagram

[redacted]

Standard Bibliographic Page

1. Report No.
NASA TM-87652

2. Government Accession No.

3. Recipient's Catalog No.

4. Title and Subtitle
OPERATION OF THE HP2250 WITH THE HP9000 SERIES 200
USING PASCAL 3.0

5. Report Date
February 1986

6. Performing Organization Code
506-43-81

7. Author(s)
John Perry (PRC Kentron, Inc.) and C. W. Stroud (LaRC)

8. Performing Organization Report No.

9. Performing Organization Name and Address
NASA Langley Research Center
Hampton, VA 23665-5225

10. Work Unit No.

11. Contract or Grant No.

12. Sponsoring Agency Name and Address
National Aeronautics and Space Administration
Washington, DC 20546-0001

13. Type of Report and Period Covered
Technical Memorandum

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

A computer program has been written to provide an interface between the HP Series 200 desktop computers, operating under HP Standard PASCAL 3.0, and the HP2250 Data Acquisition and Control System. PASCAL 3.0 for the HP9000 desktop computer gives a number of procedures for handling bus communication at various levels. It is necessary, however, to reach the lowest possible level in PASCAL to handle the bus protocols required by the HP2250. This makes programming extremely complex since these protocols are not documented. The program described herein solves those problems and allows the user to immedately program, simply and efficiently, any measurement and control language (MCL/50) application with a few procedure calls. The complete set of procedures is available on a 5 1/4" diskette from Cosmic. Included in this group of procedures is an Exerciser which allows the user to exercise his HP2250 interactively. The Exerciser operates in a fashion similar to the Series 200 operating system programs, but is adapted to the requirements of the HP2250.

The requirements for linking to a user's programs are described in detail when the diskette is used as received. The procedure for communicating with the HP2250 is very straightforward, once a user's program has been debugged and compiled. The programs on the diskette and the user's manual assume the user is acquainted with both the MCL/50 programming language and HP Standard PASCAL 3.0 for the HP series 200 desktop computers.

17. Key Words (Suggested by Authors(s))
HP2250
HP9000
PASCAL 3.0

18. Distribution Statement
[redacted]

Subject Category 59

19. Security Classif.(of this report)
Unclassified

20. Security Classif.(of this page)
Unclassified

21. No. of Pages
34

22. Price

[redacted]