HEWLETT-PACKARD

Networking HP-71s to a Supervisory Computer

Summary

There are two primary applications for networking HP-71s to a supervisory computer in a low-cost manufacturing system:

- Statistical process monitoring, where the HP-71 is an intelligent terminal
- Distributed instrument control, where the HP-71 is an intelligent low-cost controller

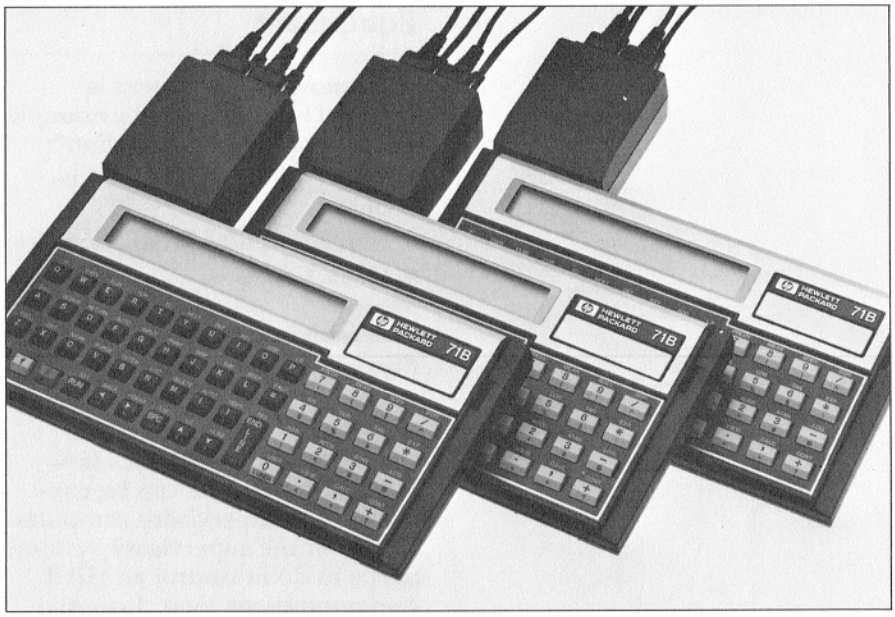
Networking HP-71s to a supervisory computer provides several advantages in a manufacturing setting:

- **Faster information flow.** Networking provides fast, accurate data transfer, eliminating the information loss inherent in verbal and paper communications. (For more information on computerized production information flow, see HP application notes 5954-1316 "Using the HP-71 for Production Process Monitoring" and 5954-1317 "Low-Cost Production Testing and Instrument Control.")
- **Network overview.** Data from several HP-71s can be combined to give an overall picture of a manufacturing line's performance. Trends can be identified and historical data gathered.
- **Information sharing.** Up to 30 HP-71s can be networked to a supervisory computer. An HP-71/HP-IL system is a small local network and is not intended to be a factory-wide networking system in itself. (The supervisory computer can pass information to the next level through a factory-wide Local Area Network.) The optimum number of HP-71s connected to a supervisory computer is determined by the application. In a network, each HP-71 is linked as a slave device on the HP-IL communications loop. The network is controlled by a supervisory computer and information is passed in one of two ways:
  1. The supervisory computer polls the HP-71 for information to be sent.
  2. The HP-71 sends a request for service to the supervisory computer.

Two considerations are important when networking to a supervisory computer. First, the network's communications scheme must not interfere with the HP-71's performance of its primary task. Each HP-71 slave device has a primary function and, when communicating with its supervisory computer, the slave device is not performing its primary function. When a test station is waiting for a new part to test, an assembly worker is assembling the next part, or there are no statistics to be calculated at the process monitoring station, the slave devices are idle. The communications scheme should allow slave devices to talk to their supervisory computer during these idle times.

A second consideration is the maximum number of slave devices to be connected to the supervisory computer. This number is determined by how much data must be passed (that is, the time needed to pass the data) and the amount of idle time. If the data transfer time is short and the idle time is long, the supervisory computer can handle many slave devices with little delay. If the transfer time is long and the idle time is short, a network containing only a few slaves will cause noticeable delays.

To reduce these delays, data should be defined and moved in short blocks. While an HP-71 can work at up to 4K bytes per second, valuable time is lost each time a new transmission is started. For example, if a data transmission takes five seconds, an operator will have to wait for five seconds to start a new procedure. If, on the other hand, the transmission sequence takes only ½ second, there will be little noticeable delay.



Equipment

The following equipment is required to implement the example detailed below in "Operation":

- Four HP-71 Handheld Computers
- Four HP 82401A HP-IL Modules
- HP-IL Cables

Operation

The following example of a simple network illustrates how HP-71 slave devices can be connected to a supervisory computer and what the supervisory system needs to do to control an HP-IL communications loop. Extensive commentary accompanies the complete code listings, so system programmers can use this information to implement networks for their own specific applications.

While the HP-71 and HP-IL are capable of functions beyond those shown, these programs provide a core of networking functions from which specific application solutions can evolve. The programming techniques used here will help you avoid pitfalls which might result in unreliable systems with hard-to-find bugs.

To avoid problems, follow the examples closely, particularly in the area of HP-IL network operations. Each specific application, however, will require program modifications to accomplish the intended purpose.

The two programs are first loaded into the supervisory controller HP-71. The controller runs the "CONTROL" program to load the "PROG" program into each of the HP-71 slave devices on its network. During initialization, the controller displays the address, identification, and status of each slave device on the loop. The controller also provides an address for each slave device and sets each device's time and date to match its own.

In our example, up to nine slave devices – normally, but not necessarily, HP-71s – can be connected to the controller on an HP-IL network loop. An actual implementation would not be limited to nine devices. If slave devices other than HP-71s are included, the controller will ignore them during system initialization.

After initialization, a digit key on the controller may be pressed to send a sequence-numbered message to the slave device whose address corresponds to the key. Pressing any key on the slave device will send a similar message back to the controller. When the controller or slave device receives a message, it checks the sequence number, and then beeps and displays an error message if the number is not correct. When the program is completed, the zero key on the controller is pressed to turn off the entire network.

The CONTROL Program

The following listing should be placed in a file named CONTROL. It runs in the controller HP-71 and manages the network.

Line #	Program Statement
1000	ENDLINE CHR$(32)&CHR$(13)&CHR$(10) @ DELAY 1
1010	RESET HP-IL @ STANDBY 2 @ RESTORE IO @ REMOTE
1020	FOR A=1 TO 9 @ T(A)=-1 @ SEND LISTEN A @ NEXT A @ A=1 @ SEND DDL 1
1030	I$=DEVID$(A) @ I=DEVAID(A) @ IF I$="" AND I=-1 THEN LOCAL @ GOTO 1100
1040	DISP A;"%";STR$(I);" ";I$; @ IF I#3 THEN DISP @ A=A+1 @ GOTO 1030
1050	S=BINEOR(SPOLL(A),1) @ CLEAR :A @ OUTPUT :A ; "REQUEST";S @ WAIT .5
1060	IF SPOLL(A)#S THEN CLEAR :A @ DISP " NOT READY" @ A=A+1 @ GOTO 1030
1070	OUTPUT :A ;"SETTIME";TIME;" @ SETDATE";DATE;" @ A=";A
1080	OUTPUT :A ;"PURGE PROG @ RUN PROG:LOOP" @ COPY PROG TO :A
1090	DISP " RUNNING" @ T(A)=0 @ R(A)=0 @ A=A+1 @ GOTO 1030
1100	T(A)=-2 @ I=READINTR @ ENABLE INTR 8 @ ON INTR GOSUB 2000
1110	DISP "CONTROLLER READY"
1120	K$=KEY$ @ IF K$="0" THEN 3000
1130	SEND IDY @ IF K$<"1" OR K$>"9" THEN 1120
1140	A=VAL(K$) @ IF T(A)<0 THEN DISP "DEVICE NOT READY" @ GOTO 1110
1150	T(A)=T(A)+1 @ 0$="CONTROL#"&STR$(A)&" MESSAGE "&STR$(T(A))
1160	SEND UNL LISTEN A DDL 0 @ OUTPUT :A ;0$ @ DISP 0$ @ GOTO 1110
2000	I=READINTR @ I=1
2010	IF T(I)<0 OR SPOLL(I)#226 THEN 2040
2020	ENTER :I ;I$ @ DISP I$
2030	R(I)=R(I)+1 @ IF R(I)#VAL(I$[LEN(I$)-2]) THEN 2060
2040	IF T(I)>-2 THEN I=I+1 @ GOTO 2010
2050	DISP "CONTROLLER READY" @ ENABLE INTR 8 @ RETURN
2060	BEEP @ DISP "MESSAGE ERROR" @ STOP
3000	REMOTE @ SEND UNL @ FOR A=1 TO 9 @ IF T(A)>=0 THEN SEND LISTEN A
3010	NEXT A @ SEND DDL 1 MTA DATA "BYE" EOL @ LOCAL @ BYE @ STOP

CONTROL Program Comments

Initialization:

Lines 1000-1020 perform several initialization functions which are necessary before the program can take control of the network. The ENDLINE sequence in line 1000 protects against a memory-lost error which could occur when HP-IL modules are used with revision A firmware. This sequence is not needed if the module contains revision B code. The display delay is set for one second so you can see the messages briefly as they are transmitted and received. In an actual application, the delay would always be set to 0,0 or 0,INF so that the display doesn't slow system operation.

Line 1010 initializes the HP-IL loop, assigns addresses to all slave devices, and places the loop in remote mode in preparation for sending commands to those devices. The standby (time-out) parameter is set at two seconds to speed the controller's detection of a broken or hung loop. Although a 2-second time-out is sufficient for an HP-71 with 16K bytes of RAM, it might not be long enough if there is additional RAM. If an HP-71 slave device has a 96K-byte plug-in, for example, the standby parameter should be set higher, perhaps to five seconds, to allow for the memory-lost condition without causing a time-out error. In the case of an HP-71 slave device which has been given the memory-clearing command (INIT:3), a significant amount of time will elapse after the command is received.

In line 1020, the variable A is used as a device address and array T is initialized to −1 to designate all devices on the loop as either non-existent or as non-slave HP-71s. Each HP-71 slave is then redesignated a listener, the address variable is reset to 1, and the DDL 1 command is sent.

If an HP-71 slave device is in remote idle mode, it may interpret data received from the controller as a command. If the slave is running a program, waiting for keyboard input, or an interrupt is pending, the data will be held in the slave device's input buffer and will not be executed until the slave is idle. The slave device will become idle when the program is suspended or terminated, and when any pending interrupt is serviced or cancelled. The ending line-feed character will not be retransmitted but will be held in the slave device buffer until the device is idle and the command is executed. This will cause a time-out and may cause an error message in the controller display.

Polling:

Lines 1030-1090 poll the slave devices on the HP-IL network to find the HP-71 slaves to be loaded with the DEVICE program.

Line 1030 checks to see if the end of the HP-IL loop has been reached. If a device does not respond with either a slave or a device ID, the CONTROL program assumes that no device exists and that the end of the loop has been reached. Local mode is then set and the code exits to the next section (line 1100). Some HP-IL devices lack one or the other of the two IDs, so both must be checked to determine that there are no more devices on the loop.

Line 1040 displays the device address, the device and slave IDs, and checks to see if the device is an HP-71 accessory (accessory ID = 3). If the device is not an HP-71 accessory, the display is terminated, the address variable is incremented, and the program returns to line 1030 to poll the next device.

Lines 1050 and 1060 determine whether an HP-71 slave device is busy, or whether it is idle and can accept remote-mode commands. This is done by reading the device status with the SPOLL command and then inverting the least significant bit with an Exclusive OR operation. The slave device buffer is cleared in case any data was left from a previously unexecuted command, and the device is then instructed to change its status to the modified value (REQUEST command). Assuming that the HP-IL loop is intact, the OUTPUT command will not cause an error, even if the device holds the line-feed character and the controller times out. This behavior will be discussed more fully in the section on network error recovery (page 8).

After a ½ second pause to give the slave device time to execute the REQUEST command, the status SPOLL (A) is read again and, if it agrees with the modified value S, the program proceeds. Otherwise, the slave device has not executed the command, is busy, and so is unable to receive remote-mode commands. In this case, the slave device buffer is cleared (the REQUEST command is probably being held there); the controller display informs you that the device is not ready; the device address is incremented; and the CONTROL program returns to line 1030 to poll the next device.

Lines 1070-1090 command the slave device to set its time and date to match those of the controller, and give the device the controller's address by initializing variable A to the value of variable A in the controller. The program to be loaded into the slave devices is named "PROG." An actual application would probably use a different file name. It might also want to load multiple files into the slaves and different sets of files into different slave devices.

Line 1080 commands the slave device to purge its program file to avoid an error if that, or another file of the same name, already exists in the slave. If no file of that name exists, the PURGE command simply causes a warning message which can be ignored. Then the command RUN PROG:LOOP is sent to the slave device. This command is equivalent to a COPY:LOOP followed by a simple RUN PROG command. The controller then satisfies the slave device command by executing its own COPY to load the file into the device. **Note that this program does not check to see if there is enough memory in the slave device to contain the program.**

Line 1090 indicates via the display that the HP-71 slave device is running the program; initializes the T and R array variables to zero; increments the device address; and returns to line 1030 to poll the next slave device on the HP-IL loop. The T and R arrays hold the sequence number of the last transmitted and received messages.

Application:

Lines 1100-1160 contain the main application program. In a real application, the system controller might have several other duties in addition to monitoring and controlling network communications. Supervisory displays and reports will need to be generated, and perhaps information will need to be transmitted to a mainframe computer for management purposes.

To make this programming task as modular as possible, all network communications are interrupt-driven so application routines need not be concerned with receiving messages from other devices on the loop. (This function is handled by the interrupt subroutine beginning at line 2000.) When application routines need to send a command or message to a device, they simply do so immediately. When slave devices send data to their controller, they generate service requests which interrupt the CONTROL program. The program then branches to the interrupt subroutine in lines 2000-2060 to receive the message. By so doing, application routines do not need to periodically poll devices to see if they have data to send.

In the two demonstration programs, messages are simple string variables. In real applications they might also include numeric variables and even complete data files. In the demonstration examples, the controller and slave devices are tightly coupled; each line of data generates a network message. This is not necessary and may not be desirable in real situations. The power and memory in an HP-71 allow it to operate independently of the network for a relatively long period of time before sending the controller a large block of data. This provides for reconfiguration or additional flexibility, since the slave devices can continue to operate in this mode even when the network is not functional due to a device failure.

Line 1100 stores a special flag in the T array variable to indicate the last device on the loop (−2), and initializes the interrupt system in the controller. READINTR clears any previous interrupts from the interrupt-cause byte. ENABLE INTR 8 sets the interrupt mask to allow only service requests to cause interrupts. The ON INTR statement specifies the location of the interrupt-service subroutine. Not that the interrupt-cause byte sets flags which indicate which of the possible interrupt conditions has occurred since the last time it was read. After it is read, it is cleared. The interrupt-enable mask defines which of the conditions in the interrupt-cause byte will precipitate an interrupt branch. The mask is cleared when a program begins and ends and when the interrupt branch is taken, so it needs to be restored at the end of the interrupt-service routine. ON INTR is the "master switch" that allows an interrupt branch to occur and specifies the action to be taken. (An OFF INTR statement disallows *any* interrupts). Interrupts are automatically disallowed when a program ends normally, but not if the program is suspended. After the CONTROLLER READY message is displayed, **Line 1120** scans the keyboard. If the zero key is pressed, the program branches to the section which terminates the demonstration by turning off the entire system.

Line 1130 sends an IDY after each key scan so that service requests from the slave devices can be received. If the HP-IL loop were allowed to be completely idle, the devices could not indicate a need for service. If the pressed key is not a digit, it is ignored, and the keyboard is scanned again. When a valid key is received, the corresponding transmit sequence number for that slave device is checked. If the sequence number is negative, the device is not running the demonstration program, so the key is ignored and the program scans the keyboard again after displaying the appropriate message. When a key corresponding to an active HP-71 device is received, the transmit sequence number is incremented and the message is placed in the output string variable (O$).

Device Addressing:

Line 1160 sends a listen address and DDL 0 command to the slave device and then sends the output string on to the network. The message is also displayed, and the routine returns to scan the keyboard again. The listener condition is set in the slave device for a number of reasons, most of which do not result in any data actually being sent to the device. TRIGGER, CLEAR, and SEND DDL'n' all cause the device listener bit to be set. If the device-interrupt routine then tries to execute an ENTER statement to receive data, the program will hang because no data is in the buffer, nor is any data likely to come. Because of this, the controller should give the slave device some positive indication that data other than the listen-address command will be sent. The easiest way to do this is to have the slave device interpret a device-dependent command as a signal to receive data. This is the purpose of the DDL 0 command before the controller sends the output string.

Interrupt Subroutines:

Line 2000 is the beginning of the Interrupt Service subroutine. The interrupt-cause byte should be read (and hence, cleared) at the beginning of the subroutine so that any interrupts which come in during subroutine execution will not be lost. In this case, the bits in the interrupt byte do not need decoding since only one interrupt condition is enabled (Service Request). An address variable (I) is initialized, and the loop beginning in **line 2010** reads the status of each of the devices on the network to determine which device requested service.

Note that the variable names in this subroutine should be different from those in the main program to avoid conflicts and unwanted interactions (C9 instead of C, for example).

Line 2020 reads and displays the message(s). **Line 2030** increments the expected sequence number and compares it with the received sequence number. If they do not agree, the program beeps, displays an error message and stops.

Line 2040 checks to see if all the devices on the loop have been polled (-2 is the end of the loop), increments the address, and goes back to line 2010 to check the next device. When all the devices have been polled, the subroutine places the "ready" message in the display, restores the interrupt mask, and returns. An interrupt can occur at the beginning of any line in the CONTROL program. Therefore, a RETURN statement needs to be included in the same program line as an ENABLE INTR statement so that interrupts don't overflow the return stack if they occur during subroutine execution.

Reset and Shutdown:

Line 2050 resets the controller after an interrupt and places the "Controller Ready" message in the display. If an interrupt error occurs, **line 2060** gives an audible "beep," displays "Message Error," and halts the program.

The shutdown code segment beginning in **line 3000** sends the listen address and DDL 1 commands to all the HP-71s which were running the DEVICE program, and then turns them off by sending the BYE command. The controller then returns the loop to local mode and turns itself off.

The DEVICE Program

This program for the slave unit(s) is similar in many ways to the CONTROL program and should be entered into the controller in a file named PROG. The controller will copy this program into each device on the HP-IL loop.

Line #	Program Statement
1000	ENDLINE CHR$(32)&CHR$(13)&CHR$(10) @ DELAY 1 @ T=0 @ R=0
1010	REQUEST 161 @ I=READINTR @ ENABLE INTR 17 @ ON INTR GOSUB 2000
1020	DISP "DEVICE";A;"READY"
1030	IF KEY$="" OR O$#"" THEN 1030
1040	T=T+1 @ O$="DEVICE "$STR$(A)&" MESSAGE "&STR$(T) @ REQUEST 226 @ GOTO 1020
2000	I=READINTR
2010	IF BIT(I,4) THEN OUTPUT :LOOP ;O$ @ REQUEST 161 @ DISP O$ @ O$=""
2020	IF NOT BIT(I,0) THEN 2050
2030	D=READDDC @ IF D=32 THEN 2040 ELSE IF D=33 THEN 2060 ELSE 2050
2040	ENTER :LOOP ;I$ @ DISP I$ @ R=R+1 @ IF R#VAL(I$[LEN(I$)-2]) THEN 2070
2050	DISP "DEVICE";A;"READY" @ ENABLE INTR 17 @ RETURN
2060	REQUEST 163 @ DISP "DONE" @ STOP
2070	BEEP @ REQUEST 163 @ DISP "MESSAGE ERROR" @ STOP

DEVICE Program Comments

Initialization:

Line 1000 performs the same function as Line 1000 in the CONTROL program and also initializes the transmit and receive sequence numbers.

Line 1010 sets the device status (161 indicates that the device is ready to receive data) and enables interrupts. The interrupt mask is set to recognize the "talker" and device-dependent command interrupts.

Lines 1020-1040 contain the application section of the program. The ready message is displayed and the keyboard is scanned. If no key was pressed or if a message is waiting to be sent, the keyboard is scanned again. If a key is pressed and the output string is empty, the transmit sequence number is incremented and the message is placed in the output string. Service is requested from the controller (226 indicates the device is ready to send data and causes the service bit to be set), and the routine goes back to scan the keyboard and wait for the controller to take the data.

Interrupts:

Line 2000 is the beginning of the interrupt subroutine. If an interrupt occurs, the interrupt-cause byte is read and cleared, and the particular bit set in that byte indicates the type of interrupt. Bit 4 is the "talker" bit and tells the device the controller is ready to receive its message. The device sends the message, returns its status to normal, displays the transmitted message to the user, and clears the output string so the application routine can send another message.

Line 2020 checks to see if the device-dependent command-interrupt bit is set. If so, the subroutine reads the command number into the variable D. This is necessary to preserve the command number because READDDC also clears the command number byte to −1. If the command is DDL 0, the controller is sending a message and the device reads in the data in **line 2040,** displays it, increments the expected sequence number, and compares it to the received sequence number. If the sequence numbers differ, the device beeps, displays an error message, and stops after setting its status to 163 (not ready to send or receive data). If the command is DDL 1, the device program is stopped after displaying a message for the user and setting the device status to 163. If a message is correctly received, or if the command is not DDL 0 or DDL 1, **line 2050** restores the device display and interrupt mask and returns to the application routine.

In a real application, it may be necessary for the DEVICE program to perform some time-critical tasks to turn off interrupts. It is important to re-enable interrupts as soon as possible, because the controller may send a message to a slave device and be hung waiting for the device to accept it. A time-out and a system error could occur if the controller does not allow for this possibility. Alternatively, the controller might be required to check the slave device status before sending any message, and the device could set a "not ready" status when it becomes necessary to disable the interrupts.

Please note that in both the CONTROLLER and DEVICE programs, simple subroutines are used for interrupt servicing. This means that variables used in subroutines may be shared with variables of the same names in the application sections of these programs. To prevent unexpected interaction and difficult-to-find bugs, use obscure names in the interrupt subroutines so accidental duplication of names is less likely. For example, you might use I9 instead of I.

Network Error Recovery

In most practical situations various system errors can and do occur. An HP-71 network must be capable of recognizing and recovering from these errors. The importance and level of error recovery depends upon the particular application, so only rough guidelines and procedures can be given here. Two types of errors can occur on an HP-IL network:

- The HP-IL loop may be broken so that frames cannot return to their sourcing device. This could be a physical problem such as a cable break or a loose connector, or it may be something more subtle such as no power on one of the devices in the loop. **The system will not function at all when this type of error occurs.**
- The second type of error is a device problem. The DEVICE program may have been accidentally suspended, or a "bug" or operator error could cause the program to stop accepting data from the controller. In this case, the controller can usually recover control of the remainder of the system and continue to service the other devices in the HP-IL loop.

When either of these errors occurs, the controller should notify the system operator to take corrective action. After the action has been taken, the controller should resume control of the entire system or of the offending slave device.

Data Loss:

Loss of data is unacceptable in many applications. When an error occurs, the message being transmitted may be garbled or lost. To prevent such loss, both the controller and the device(s) on the loop must be enabled to retransmit the previous message. When system operation has been restored after an error, the controller can determine if the previous message was lost and can then retransmit or request retransmission of the message.

To determine that a message has been lost, the controller must have some specific capabilities. Also, the receiving unit must ignore partial or garbled messages. Two ways to satisfy these requirements are to define fixed-length messages or to use a special flag character as the message terminator. In addition, messages may require some kind of alternating sequence flag similar to the sequence numbers used in our example. Then the controller can determine whether or not a message was completed prior to an error, or whether it was lost when the error occurred by checking the received or transmitted message flag in the device.

Errors:

When an error occurs, a device HP-71 does not time-out. It also ignores the STANDBY command parameter and waits indefinitely for an HP-IL operation to complete. When the controller is satisfied that the network is intact after an error, its first order of business is to make sure the DEVICE program can proceed and that the program is functioning normally.

If the device is executing an OUTPUT statement when the error occurs, the controller can clear the DEVICE program by reading the rest of the data from the device and then discarding that data. The controller might receive nothing or a complete message, depending on when the error occurs. If the device is executing an ENTER statement when the error occurs, it will usually wait for a line-feed character to terminate the statement before proceeding. The controller can then send a line-feed character to the device to get the DEVICE program to continue. This may leave the line-feed character in the device buffer if the device is not waiting at an ENTER statement, so it is necessary to follow the line-feed character with a device clear command on HP-IL to cover this possibility. When the DEVICE program is again running normally, the controller can check the message sequence flag or perform whatever higher-level actions are required to recover from a possible loss of data.

Program Suspension:

Although an HP-71 device is not concerned with time-out errors, it does respond to two other types of errors that can occur on the HP-IL loop. These errors can cause program suspension if precautions are not taken. For example, when the HP-IL loop is broken in the middle of a frame, a "talker" device can receive a garbled frame which appears to the device as a protocol violation. In the device, these types of errors should be ignored so that the program will continue to run (discarding the message, of course) and allow the controller to handle the situation and recover. **It is usually undesirable to ignore DEVICE program errors** but it can be done by using ON ERROR GOSUB prior to ENTER LOOP or OUTPUT LOOP. The target error subroutine would simply be a RETURN, so the program will continue despite the error. OFF ERROR should probably be executed immediately after ENTER or OUTPUT. ON ERROR could be left active during the entire DEVICE program rather than enabling and disabling it for each input/output operation, but as noted above, DEVICE program errors should not be ignored.

The HP-71 as System Controller:

An HP-71 is not normally used as the system controller, but there are some situations where this might be desirable. Two characteristics of the OUTPUT and ENTER statements in an HP-71 controller affect its ability to recover the network from errors. Under certain conditions, the OUTPUT statement can time-out without causing an error in the program. The data which was to have been sent to the device is sent instead to the controller's display. When this happens, there will be no indication that the transmission has failed, so it becomes necessary for the controller to verify in some other way that the message was received. This can be done by checking the slave device's received-message sequence flag after each transmission.

When an HP-71 controller executes an ENTER statement and a slave device fails to supply the proper termination sequence (usually a line-feed character), the controller may hang and not time out. This also may happen when the HP-IL loop is broken, preventing the termination sequence from reaching the controller. Once a system is operating relatively free from bugs in the DEVICE program this seldom happens, but you should be aware of such a possibility. If it does happen, first correct the network or slave device problem and then press the controller's [ATTN] key twice to suspend the CONTROL program. Then resume the CONTROL program at the point where it recovers from a network error, as described above.

In general, there is no way for an HP-71 controller to determine if an HP-71 slave device is busy other than to attempt to execute a remote-mode command and either trap any error that occurs or verify that the command has been executed. Unless the controller verifies that the command was executed, it might assume that the slave device is idle and ready to receive further commands. For example, it is possible that an ENTER statement might be executed when an HP-71 slave device is running a program. When the controller then sends a remote-mode command, the data will be taken by the ENTER statement in the DEVICE program but will not be executed as a command. No time-out or error will occur.

The controller also has no way to cancel a busy condition in a slave device so the device can be prepared to receive commands. If the controller determines that a slave device is busy, manual intervention at the device is necessary; a running program must be suspended and the STOP command may be needed to cancel a pending interrupt. Consequently, it may be important to design the DEVICE program so that the controller can terminate it and, subsequently, send commands to the slave device.

In this example, the DEVICE program is designed to terminate when a DDL 1 command is received. The CONTROL program sends DDL 1 as part of the network set-up to gain control, even if the DEVICE program is running. There are several ways to implement this function, but this method is simple and illustrates how other slave-dependent commands can be used to execute functions in the DEVICE program.

If the CONTROL or DEVICE programs are inadvertently suspended, network errors are likely to occur, so program suspensions should be prevented. Though not used here as part of their initialization sequences, preventive programs can disable the ATTN key and other key sequences that could cause programs in the network to malfunction.

To disable the ATTN key, execute this statement:

```
POKE "2F441","F"
```

To reenable the ATTN key, execute this statement:

```
POKE "25441","0"
```

Disabling the ATTN key prevents suspension of a running program, provided the program does not use the INPUT statement. Since most programs include the INPUT statement, inadvertent suspension must be prevented by redefining the ATTN key in addition to executing the POKE statement above.

To define the ATTN key to a null character, execute the following statement:

```
DEF KEY "43","";
```

Because user mode must be active for the key definition to be effective, the USER ON statement is also executed. Since the USER and 1USER keys are active while waiting for keyboard response at an INPUT statement, they should also be redefined to prevent deactivation of user mode. Appropriate statements might be:

```
DEF KEY "109",""; and DEF KEY "165","";
```

While these procedures protect against inadvertent suspension of the program, there should be a way for the system controller or operator to terminate the program. For example, a program allowing correct termination might interpret a particular sequence as a STOP command. In the CONTROL program in this example, the zero key serves this function. Should a program hang and refuse to respond to the proper terminating sequence, the HP-71 initialization sequence can be used to recover. The ON and / keys are pressed at the same time and, when the INIT:1 prompt appears in the display, the ENDLINE key is pressed. This initialization sequence cannot be disabled, but it is unlikely to be executed inadvertently.

For additional information about Hewlett-Packard's handheld computers in manufacturing solutions, call the HP office nearest you and ask for your technical computer or instrument representative. Local HP sales offices are listed in the white pages of your telephone book.

Hewlett-Packard
Handheld Computer and
Calculator Operation
1000 N.E. Circle Blvd.
Corvallis, Oregon 97330

Printed in USA. 10/85
5954-1312 WCP 5.0