4

Using HP SICL with HP-IB

Using HP SICL with HP-IB

The HP-IB interface (Hewlett-Packard Interface Bus) is Hewlett-Packard's implementation of the IEEE 488.1 Bus. Other IEEE 488 versions include GPIB (General Purpose Interface Bus) and IEEE Bus. GPIB and HP-IB are both used in the discussions and examples in this chapter. The HP-IB related SICL functions have the string GPIB embedded in the function name.

This chapter explains how to use SICL to communicate over HP-IB. In order to communicate over HP-IB, you must have loaded the HPIB fileset during the system installation. See the *HP I/O Libraries Installation and Configuration Guide for HP-UX* for information.

This chapter describes in detail how to open a communications session and communicate with HP-IB devices, interfaces, or controllers. The example programs shown in this chapter are also provided in the `/opt/sicl/share/examples` directory on HP-UX 10, or the `/usr/pil/examples` directory on HP-UX 9.

This chapter contains the following sections:

- Creating a Communications Session with HP-IB
- Communicating with HP-IB Devices
- Communicating with HP-IB Interfaces
- Communicating with HP-IB Commanders
- Summary of HP-IB Specific Functions

Creating a Communications Session with HP-IB

Once you have determined that your HP-IB system is setup and operating correctly, you may want to start programming with the SICL functions. First you must determine what type of communication session you need. The three types of communications sessions are device, interface, and commander.

Communicating with HP-IB Devices

The device session allows you direct access to a device without worrying about the type of interface to which it is connected. The specifics of the interface are hidden from the user.

Addressing HP-IB Devices

To create a device session, specify either the interface `symbolic name` or `logical unit` and a particular device's address in the *addr* parameter of the `iopen` function. The interface `symbolic name` and `logical unit` are defined during the system configuration. See the *HP I/O Libraries Installation and Configuration Guide for HP-UX* for information on these values.

The following are example HP-IB addresses for device sessions:

`hpib,7`	A device address corresponding to the device at primary address 7 and symbolic name `hpib`.
`hpib,3,2`	A device address corresponding to the device at primary address 3, secondary address 2, and symbolic name `hpib`.
`hpib,9,0`	A device address corresponding to the device at primary address 9, secondary address 0, and symbolic name `hpib`.

Note The above examples use the default `symbolic name` specified during the system configuration. If you want to change the name listed above, you must also change the `symbolic name` or `logical unit` specified during the configuration. The name used in your SICL program must match the `logical unit` or `symbolic name` specified in the system configuration. Other possible interface names are `GPIB`, `gpib`, `HPIB`, etc.

SICL supports both primary and secondary addressing on HP-IB interfaces.

Remember that the primary address must be between 0 and 30 and that the secondary address must be between 0 and 30. The primary and secondary addresses correspond to the HP-IB primary and secondary addresses.

Note If you are using an HP-IB Command Module to communicate with VXI devices, the secondary address must be specified to select a specific instrument in the cardcage. Secondary addresses of 0, 1, 2, . . .31 correspond to VXI instruments at logical addresses of 0, 8, 16, . . . 248, respectively.

The following is an example of opening a device session with an HP-IB device at bus address 16:

```
INST dmm
dmm = iopen ("hpib,16");
```

HP SICL Function Support with HP-IB Device Sessions

The following describes how some SICL functions are implemented for HP-IB device sessions.

`iwrite`	Causes all devices to untalk and unlisten. It then sends this controller's talk address followed by unlisten and then the listen address of the corresponding device session. Then it sends the data over the bus.
`iread`	Causes all devices to untalk and unlisten. It sends an unlisten, then sends this controller's listen address followed by the talk address of the corresponding device session. Then it reads the data from the bus.
`ireadstb`	Performs a GPIB serial poll (SPOLL).
`itrigger`	Performs an addressed GPIB group execute trigger (GET).
`iclear`	Performs a GPIB device clear (DCL) on the device corresponding to this session.

HP-IB Device Session Interrupts

There are no device-specific interrupts for the HP-IB interface.

HP-IB Device Sessions and Service Requests

HP-IB device sessions support Service Requests (SRQ). On the HP-IB interface, when one device issues an SRQ, the library will inform *all* HP-IB device sessions that have SRQ handlers installed. (See `ionsrq` in Chapter 10.) This is an artifact of how HP-IB handles the SRQ line. The interface cannot distinguish which device requested service. Therefore, the library acts as if all devices require service. Your SRQ handler can retrieve the device's **status byte** by using the `ireadstb` function. It is good practice to ensure that a device isn't requesting service before leaving the SRQ handler. The easiest technique for this is to service all devices from one handler.

The data transfer functions work only when the HP-IB interface is the Active Controller. Passing control to another HP-IB device causes the interface to lose active control.

HP-IB Device Session Example

The following example illustrates communicating with an HP-IB device session. This example opens two HP-IB communications sessions with VXI devices (through a VXI Command Module). Then a scan list is sent to a switch, and measurements are taken by the multimeter every time a switch is closed.

```
/* hpibdev.c
   This example program sends a scan list to a switch and
   while looping closes channels and takes measurements.*/
#include <sicl.h>
#include <stdio.h>

main()
{
   INST dvm;
   INST sw;

   double res;
   int i;

   /* Print message and terminate on error */
   ionerror (I_ERROR_EXIT);

   /* Open the multimeter and switch sessions */
   dvm = iopen ("hpib,9,3");
   sw = iopen ("hpib,9,14");
   itimeout (dvm, 10000);
   itimeout (sw, 10000);

   /*Set up trigger*/
   iprintf (sw, "TRIG:SOUR BUS\n");

   /*Set up scan list*/
   iprintf (sw,"SCAN (@100:103)\n");
   iprintf (sw,"INIT\n");

   for (i=1;i<=4;i++)
   {
      /* Take a measurement */
      iprintf (dvm,"MEAS:VOLT:DC?\n");

      /* Read the results */
      iscanf (dvm,"%lf",&res);

      /* Print the results */
      printf ("Result is %f\n",res);

      /*Trigger to close channel*/
      iprintf (sw, "TRIG\n");
   }
   /* Close the multimeter and switch sessions */
   iclose (dvm);
   iclose (sw);
}
```

Communicating with HP-IB Interfaces

Interface sessions allow you direct low-level control of the interface. You must do all the bus maintenance for the interface. This also implies that you have considerable knowledge of the interface. Additionally, when using interface sessions, you need to use interface specific functions. The use of these functions means that the program can not be used on other interfaces and, therefore, becomes less portable.

Addressing HP-IB Interfaces

To create an interface session on your HP-IB system, specify either the interface `symbolic name` or `logical unit` in the *addr* parameter of the `iopen` function. The interface `symbolic name` and `logical unit` are defined during the system configuration. See the *HP I/O Libraries Installation and Configuration Guide for HP-UX* for information on these values.

The following are example HP-IB interface addresses:

`hpib`	An interface symbolic name.
`hpib2`	An interface symbolic name.
7	An interface logical unit.

Note The above examples use the default `symbolic name` specified during the system configuration. If you want to change the name listed above, you must also change the `symbolic name` or `logical unit` specified during the configuration. The name used in your SICL program must match the `logical unit` or `symbolic name` specified in the system configuration. Other possible interface names are `GPIB`, `gpib`, `HPIB`, `IEEE488`, etc.

The following example opens a interface session with the HP-IB interface:

```
INST hpib;
hpib = iopen ("hpib");
```

HP SICL Function Support with HP-IB Interface Sessions

The following describes how some SICL functions are implemented for HP-IB interface sessions.

`iwrite`	Sends the specified bytes directly to the interface without performing any bus addressing. The `iwrite` function always clears the ATN line before sending any bytes, thus ensuring that the GPIB interface sends the bytes as data, not command bytes.
`iread`	Reads the data directly from the interface without performing any bus addressing.
`itrigger`	Performs a GPIB group execute trigger (GET) without additional addressing. This function should be used with the `igpibsendcmd` to send an `UNL` followed by the device addresses. This will allow the `itrigger` function to be used to trigger multiple GPIB devices simultaneously. Passing the `I_TRIG_STD` value to the `ixtrig` routine also causes a broadcast GPIB group execute trigger (GET). There are no other valid values for the ixtrig function.
`iclear`	Performs a GPIB interface clear (pulses IFC and REN), which resets the interface.

HP-IB Interface Session Interrupts

There are specific interface session interrupts that can be used. See `isetintr` in Chapter 10 for information on the interface session interrupts.

There are no device specific interrupts for the HP-IB interface.

HP-IB Interface Sessions and Service Requests

HP-IB interface sessions support Service Requests (SRQ). On the HP-IB interface, when one device issues an SRQ, the library will inform *all* HP-IB interface sessions that have SRQ handlers installed. (See `ionsrq` in Chapter 10.) It is good practice to ensure that a device isn't requesting service before leaving the SRQ handler. The easiest technique for this is to service all devices from one handler.

HP-IB Interface Session Examples

Checking the Bus Status

The following example program is an ANSI C program that retrieves the HP-IB interface bus status information and displays it for the user.

```
/* hpibstatus.c
   The following example retrieves and displays HPIB bus
    status information. */
#include <stdio.h>
#include <sicl.h>

main()
{
   INST id;         /* session id        */
   int rem;         /* remote enable     */
   int srq;         /* service request   */
   int ndac;        /* not data accepted */
   int sysctlr;     /* system controller */
   int actctlr;     /* active controller */
   int talker;      /* talker            */
   int listener;    /* listener          */
   int addr;        /* bus address       */

   /* exit process if SICL error detected */
   ionerror(I_ERROR_EXIT);

   /* open HPIB interface session */
   id = iopen("hpib");
   itimeout (id, 10000);

   /* retrieve HPIB bus status */
   igpibbusstatus(id, I_GPIB_BUS_REM,      &rem);
   igpibbusstatus(id, I_GPIB_BUS_SRQ,      &srq);
   igpibbusstatus(id, I_GPIB_BUS_NDAC,     &ndac);
   igpibbusstatus(id, I_GPIB_BUS_SYSCTLR,  &sysctlr);
   igpibbusstatus(id, I_GPIB_BUS_ACTCTLR,  &actctlr);
   igpibbusstatus(id, I_GPIB_BUS_TALKER,   &talker);
   igpibbusstatus(id, I_GPIB_BUS_LISTENER, &listener);
   igpibbusstatus(id, I_GPIB_BUS_ADDR,     &addr);

   /* display bus status */
   printf("%-5s%-5s%-5s%-5s%-5s%-5s%-5s%-5s\n", "REM",
       "SRQ", "NDC", "SYS", "ACT", "TLK", "LTN", "ADDR");
    printf("%2d%5d%5d%5d%5d%5d%5d%6d\n", rem, srq, ndac,
        sysctlr, actctlr, talker, listener, addr);
   return 0;
}
```

Communicating with Devices via Interface Sessions

The following example program sets up two HP-IB instruments over an interface session and has the instruments communicate with each other.

The 3 main parts of this program are as follows:

- Read the data from the scope (`get_data`).
- Print some statistics about the data (`massage_data`).
- Have the scope send the data to a printer (`print_data`).

```
/* hpibintr.c
   This program requires a 54601A digitizing oscilloscope
   or compatible) and a printer capable of printing in HP
   RASTER GRAPHICS STANDARD (e.g. thinkjet).
   This program will tell the scope to take a reading on
   channel 1, then send the data back to this program.
   Then  some simple statistics about the data is printed.
   The program then tells the scope to send the data
   directly to the printer, illustrating how the
   controller does not have to be directly involved in an
   HPIB transaction.*/

#include <stdio.h>   /* used for printf() */
#include <stdlib.h>  /* used for exit() */
#include <sicl.h>    /* SICL header file */

/* defines */
#define INTF_ADDR    "hpib"
#define SCOPE_ADDR   INTF_ADDR ",7"

/* function prototypes */
void initialize (void);
void get_data (void);
void massage_data (void);
void print_data (void);
void cleanup (void);
void srq_hdlr (INST id);

/* global data */
float pre[10];
INST scope;
INST intf;
```

```
void main() {
   ionerror(I_ERROR_EXIT);
   scope = iopen(SCOPE_ADDR);
   intf = iopen(INTF_ADDR);

   initialize();
   get_data();
   massage_data();
   print_data();
   cleanup();

   iclose(scope);
   iclose(intf);
}

void initialize() {
   /* initialize the hpib interface and scope */
   iclear(intf);
   itimeout(scope, 5000);
   itimeout(intf, 5000);
   iclear(scope);
   igpibllo(intf);
}

void get_data() {
   short readings[5000];
   int count;

   /* setup scope to accept waveform data */
   iprintf(scope, "*RST\n");
   iprintf(scope, ":autoscale\n");

   /* setup up the waveform source */
   iprintf(scope, ":waveform:format word\n");

   /* input waveform preamble to controller */
   iprintf(scope, ":digitize channel1\n");
   iprintf(scope, ":waveform:preamble?\n");
   iscanf(scope, "%,10f", pre);

   /* command scope to send data */
   iprintf(scope, ":waveform:data?\n");
```

```
  /* enter the data */
  count = 5000;
  iscanf(scope, "%#wb\n", &count, readings);
  printf ("received %d words\n", count);
}

void massage_data() {
  float vdiv;
  float off;
  float sdiv;
  float delay;
  char  id_str[50];

  vdiv  = 32 * pre[7];
  off   = (128 - pre[9]) * pre[7] + pre[8];
  sdiv  = pre[2] * pre[4] / 10;
  delay = (pre[2] / 2 - pre[6]) * pre[4] + pre[5];

  /* retrieve the scope's ID string */
  ipromptf(scope, "*IDN?\n", "%s", id_str);

   /*  print the statistics about the data */
  printf("\nOscilloscope ID:  %s\n", id_str);
  printf(" ----------  Current settings  -----------\n");
  printf("      Volts/Div = %f V\n", vdiv);
  printf("         Offset = %f V\n", off);
  printf("          S/Div = %f S\n", sdiv);
  printf("          Delay = %f S\n", delay);
}

void print_data() {
  unsigned char status;
  char    cmd[5];

  /* tell the scope to SRQ on 'operation complete'*/
  iprintf(scope, "*SRE 32; *ESE 1\n");

  /* tell the scope to print */
  iprintf(scope, ":print?; *OPC\n");
```

```
   /* tell scope to talk and printer to listen.  The listen
   command is formed by adding 32 to the device address
   of the device to be a listener.  The talk command is
   formed by adding 64 to the device address of the
   device to be a talker */
cmd[0] = 63;    /* 63 is unlisten                          */
cmd[1] = 32+1; /* printer is at address 1, make it a listener*/
cmd[2] = 64+7; /* scope is at address 7, make it a talker*/
cmd[3] = '\0';  /* terminate the string                    */

igpibsendcmd(intf, cmd, 3);

   /* set up our SRQ handler to be called when the scope
      finishes printing */
   ionsrq(scope, srq_hdlr);

   /* now, the ATN line must be set to FALSE */
   igpibatnctl(intf, 0);

/* wait for SRQ before continuing program */
   status = 0;
   while(status == 0) {
      iwaithdlr(120000L);

      /* make sure it was the scope requesting service */
      ireadstb(scope, &status);
      status &= 64;
   }

   /* clear the status byte so the scope can assert SRQ
       again if needed. */
   iprintf(scope, "*CLS\n");
}

void cleanup() {
   /* give local control back to the scope */
   ilocal(scope); }

void srq_hdlr(INST id) {
   /* this handler does nothing. we will use iwaithdlr()in
the code above to determine when the handler gets called. */
}
```

Communicating with HP-IB Commanders

Commander sessions are intended for use on HP-IB interfaces that are not active controller. In this mode, a computer that is not the controller is acting like a device on the HP-IB bus. In a commander session, the data transfer routines work only when the HP-IB interface is not active controller.

Addressing HP-IB Commanders

To create a commander session on your HP-IB interface, specify either the interface `symbolic name` or `logical unit` in the *addr* parameter followed by a comma and the string `cmdr` in the `iopen` function. The interface `symbolic name` and `logical unit` are defined during the system configuration. See the *HP I/O Libraries Installation and Configuration Guide for HP-UX* for information on these values.

The following are example HP-IB addresses for commander sessions:

`hpib,cmdr`	A commander session with the `hpib` symbolic name.
`hpib2,cmdr`	A commander session with the `hpib2` symbolic name.
`7,cmdr`	A commander session with the interface at logical unit 7.

Note The above examples use the default `symbolic name` specified during the system configuration. If you want to change the name listed above, you must also change the `symbolic name` or `logical unit` specified during the configuration. The name used in your SICL program must match the `logical unit` or `symbolic name` specified in the system configuration. Other possible interface names are `GPIB`, `gpib`, `HPIB`, etc.

The following example opens a commander session the HP-IB interface:

```
INST hpib;
hpib = iopen ("hpib,cmdr");
```

HP SICL Function Support with HP-IB Commander Sessions

The following describes how some SICL functions are implemented for HP-IB commander sessions.

`iwrite`	If the interface has been addressed to talk, the data is written directly to the interface. If the interface has not been addressed to talk, it will wait to be addressed to talk before writing the data.
`iread`	If the interface has been addressed to listen, the data is read directly from the interface. If the interface has not been addressed to listen, it will wait to be addressed to listen before reading the data.
`isetstb`	Sets the status value that will be returned on a `ireadstb` call (i.e. when this device is Serial Polled). Bit 6 of the status byte has a special meaning. If bit 6 is set, the SRQ line will be set. If bit 6 is clear, the SRQ line will be cleared.

HP-IB Commander Session Interrupts

There are specific commander session interrupts that can be used. See `isetintr` in Chapter 10 for information on the commander session interrupts.

Summary of HP-IB Specific Functions

Note Using these HP-IB interface specific functions means that the program can not be used on other interfaces and, therefore, becomes less portable.

SICL GPIB Functions

Function Name	Action
igpibatnctl	Sets or clears the ATN line
igpibbusaddr|	Change bus address
igpibbusstatus	Return requested bus data
igpibgett1delay	Retrieves the T1 delay setting on the GPIB interface
igpibllo	Sets bus in Local Lockout Mode
igpibpassctl	Passes active control to specified address
igpibppoll	Performs a parallel poll on the bus
igpibppollconfig	Configures device for PPOLL response
igpibppollresp	Sets PPOLL state
igpibrenctl	Sets or clears the REN line
igpibsendcmd	Sends data with ATN line set
igpibsett1delay	Sets the T1 delay on the GPIB interface