Using HP SICL with GPIO

GPIO is a parallel interface that is flexible and allows a variety of custom connections. Although GPIO typically requires more time to configure than HP-IB, its speed and versatility make it the perfect choice for many tasks.

This chapter explains how to use SICL to communicate over GPIO. In order to communicate over GPIO, you must have loaded the GPIO fileset during the HP I/O Libraries installation. See the *HP I/O Libraries Installation and Configuration Guide for HP-UX* for information. Also note that the GPIO related SICL functions have the string `GPIO` embedded in their names.

This chapter describes in detail how to open a communications session and communicate with an instrument over a GPIO connection. 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.

Note GPIO is *not* supported with SICL over LAN.

This chapter contains the following sections:

- Creating a Communications Session with GPIO
- Communicating with GPIO Interfaces
- Summary of GPIO Specific Functions

Creating a Communications Session with GPIO

Once you have configured your system for GPIO communications, you can start programming with the SICL functions. If you have programmed GPIO before, you will probably want to open the interface and start sending commands.

With HP-IB and VXI, there can be multiple devices on a single interface. These interfaces support a connection called a **device session**. With GPIO, only one device is connected to the interface. Therefore, you communicate with GPIO devices using an **interface session**.

Communicating with GPIO Interfaces

Interface sessions are used for GPIO data transfer, interrupt, status, and control operations. When communicating with a GPIO interface session, you specify the interface name.

Addressing GPIO Interfaces

To create an interface session on GPIO, 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 addresses for GPIO interface sessions:

`gpio`	An interface symbolic name.
`12`	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 `parallel`, `GPIO`, etc.

The following example opens an interface session with the GPIO interface:

```
INST intf;
intf = iopen ("gpio");
```

HP SICL Function Support with GPIO Interface Sessions

The following describes how some SICL functions are implemented for GPIO interface sessions.

`iwrite`, `iread`	The *size* parameters for non-formatted I/O functions are always byte counts, regardless of the current data width of the interface.
`iprintf`, `iscanf`	All formatted I/O functions work with GPIO. When formatted I/O is used with 16-bit data widths, the formatting buffers re-assemble the data as a stream of bytes. On the Series 700, these bytes are ordered: high-low-high-low... Because of this "unpacking" operation, 16-bit data widths may not be appropriate for formatted I/O operations. For `iscanf` termination, an END value must be specified using `igpioctrl`. See Chapter 10 for details.
`itermchr`	With 16-bit data widths, only the low (least-significant) byte is used.
`ixtrig`	Provides a method of triggering using either the CTL0 or CTL1 control lines. This function pulses the specified control line for approximately 1 microsecond. The following constants are defined: I_TRIG_STD Pulse CTL0 line I_TRIG_GPIO_CTL0 Pulse CTL0 line I_TRIG_GPIO_CTL1 Pulse CTL1 line
`itrigger`	Same as `ixtrig` (`I_TRIG_STD`). Pulses the CTL0 control line.
`iclear`	Pulses the P_RESET line for approximately 12 microseconds, aborts any pending writes, discards any data in the receive buffer, and resets any error conditions. Optionally clears the Data Out port, depending upon the *mode* configuration specified during the SICL configuration.

`ionsrq`	Installs a service request handler for this session. The concept of service request (SRQ) originates from HP-IB. On an HP-IB interface, a device can request service from the controller by asserting a line on the interface bus. On GPIO, the EIR line is assumed to be the service request line.
`ireadstb`	Chapter 10 says that `ireadstb` is for device sessions only. Since GPIO has no device sessions, `ireadstb` is allowed with GPIO interface sessions. The interface status byte has bit 6 set if EIR is asserted; otherwise, the status byte is 0 (zero). This allows normal SRQ programming techniques in GPIO SRQ handlers.

GPIO 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 for GPIO.

GPIO Interface Session Example

```
/* gpiomeas.c
   This program does the following:
   - Creates GPIO session with timeout and error checking
   - Signals the device with a CTL0 pulse
   - Reads the device’s response using formatted I/O
*/

#include <sicl.h>

main()
{
   INST id;       /* interface session id */
   float result;  /* data from device */

   /* log message and exit program on error */
   ionerror (I_ERROR_EXIT);

   /* open GPIO interface session, with 10-second timeout
*/
   id = iopen ("gpio");
   itimeout (id, 10000);

   /* setup formatted I/O configuration */
   igpiosetwidth (id, 8);
   igpioctrl (id, I_GPIO_READ_EOI, ’\n’);

   /* monitor the device’s PSTS line */
   igpioctrl(id, I_GPIO_CHK_PSTS, 1);

   /* signal the device to take a measurement */
   itrigger(id);

   /* get the data */
   iscanf(id, "%f%*t", &result);
   printf("Result = %f\n", result);

   /* close session */
   iclose (id);
}
```

GPIO Interrupts Example

```
/* gpiointr.c
   This program does the following:
   - Creates a GPIO session with error checking
   - Installs interrupt handler & enables EIR interrupts
   - Waits for EIR; invokes the handler for each interrupt
   - Handler checks interrupt cause & exits when EIR is
     clear
*/
#include <sicl.h>

void handler(id, reason, sec
INST id;
long reason, sec;
{
   if (reason == I_INTR_GPIO_EIR) {
      printf("EIR interrupt detected\n");

      /* Proper protocol is for the peripheral device to hold
       * EIR asserted until the controller "acknowledges" the
       * interrupt. The method for acknowledging and/or responding
       * to EIR is very device-dependent. Perhaps a CTLx line is
       * pulsed, or data is read, etc.  The response should be
       * executed at this point in the program.
       */
   }
   else
      printf("Unexpected Interrupt; reason=%d\n", reason);
}

main()
{
   INST intf;     /* interface session id */

   /* log message and exit program on error */
   ionerror (I_ERROR_EXIT);

   /* open GPIO interface session */
   intf = iopen ("gpio");
```

```
  /* suspend interrupts until configured */
  iintroff();

  /* configure interrupts */
  ionintr(intf, handler);
  isetintr(intf, I_INTR_GPIO_EIR, 1);

  /* wait for interrupts */
  printf("Ready for interrupts\n");
  while (1) {
    iwaithdlr(0);
  }

  /* iwaithdlr performs an automatic iintron().  If your program
   * does concurrent processing, instead of waiting, then you need
   * to execute iintron() when you are ready for interrupts.
   */
  /* This simplified example loops forever.  Most real applications
   * would have termination conditions that cause the loop to exit.
   */
  iclose (intf);
}
```

Summary of GPIO Specific Functions

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

Function Name	Action
igpioctrl	Sets the following characteristics of the GPIO interface:

Request	Characteristic	Settings
I_GPIO_AUTO_HDSK	Auto-Handshake mode	1 or 0
I_GPIO_AUX	Auxiliary Control lines	16-bit mask
I_GPIO_CHK_PSTS	Check PSTS before read/write	1 or 0
I_GPIO_CTRL	Control lines	I_GPIO_CTRL_CTL0
		I_GPIO_CTRL_CTL1
I_GPIO_DATA	Data Output lines	8-bit or 16-bit mask
I_GPIO_PCTL_DELAY	PCTL delay time	0-7
I_GPIO_POLARITY	Logical polarity	0-31
I_GPIO_READ_CLK	Data input latching	See *Chapter 10*
I_GPIO_READ_EOI	END termination pattern	I_GPIO_EOI_NONE or
		8-bit or 16-bit mask
I_GPIO_SET_PCTL	Start PCTL handshake	1

igpiogetwidth	Returns the current width (in bits) of the GPIO data ports.
igpiosetwidth	Sets the width (in bits) of the GPIO data ports. Either 8 or 16.

Function Name	**Action**
igpiostat	Gets the following information about the GPIO interface:

Request	Characteristic	Value
I_GPIO_CTRL	Control Lines	I_GPIO_CTRL_CTL0 I_GPIO_CTRL_CTL1
I_GPIO_DATA	Data In lines	16-bit mask
I_GPIO_INFO	GPIO information	I_GPIO_AUTO_HDSK I_GPIO_CHK_PSTS I_GPIO_EIR I_GPIO_ENH_MODE I_GPIO_PSTS I_GPIO_READY
I_GPIO_READ_EOI	END termination pattern	I_GPIO_EOI_NONE or 8-bit or 16-bit mask
I_GPIO_STAT	Status lines	I_GPIO_STAT_STI0 I_GPIO_STAT_STI1