

---

**Using SICL with GPIO**

---

## Using SICL with GPIO

This chapter shows how to open an interface communications session and communicate with an instrument over a GPIO connection. The example programs in this chapter are also provided in the `C\SAMPLES\MISC` (for C/C++) and `VB\SAMPLES\MISC` (for Visual Basic) subdirectories. This chapter includes:

- Introduction
- Using GPIO Interface Sessions

---

## Introduction

This section provides an introduction to using SICL with the GPIO interface, including:

- Selecting a GPIO Communications Session
- SICL GPIO Functions

### Selecting a GPIO Communications Session

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

#### NOTE

GPIO is *only* supported with SICL on Windows 95, Windows 98, Windows 2000, and Windows NT. GPIO is *not* supported with SICL via LAN.

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 GPIB, 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, communication with GPIO devices must be using an **interface session**.

### SICL GPIO Functions

Function Name	Action
<code>igpioctrl</code>	Sets the following characteristics of the GPIO interface:

## Using SICL with GPIO

### Introduction

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 <i>Chapter 11 - SICL Language Reference</i>
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

<b>igpiogetwidth</b>	Returns the current width (in bits) of the GPIO data ports.
<b>igpiosetWidth</b>	Sets the width (in bits) of the GPIO data ports. Either 8 or 16.

<code>igpiostat</code>	Gets the following information about the GPIO interface:
------------------------	--

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

---

## Using GPIO Interface Sessions

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

### Addressing GPIO Interfaces

To create an interface session on GPIO, specify the interface logical unit or symbolic name in the *addr* parameter of the `iopen` function. The interface logical unit and symbolic name are defined by running the `IO Config` utility. To open `IO Config`, click the `Agilent IO Libraries Control` (on the taskbar) and click `Run IO Config`. See the *Agilent IO Libraries Installation and Configuration Guide for Windows* for information on `IO Config`. Some example addresses for GPIO interface sessions are:

<code>gpio</code>	An interface symbolic name
<code>12</code>	An interface logical unit

This example opens an interface session with the GPIO interface.

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

### SICL Function Support with GPIO Interface Sessions

This section describes how some SICL functions are implemented for GPIO interface sessions.

GPIO Interface  
Sessions SICL  
Functions

<b>iwrite, iread</b>	The <i>size</i> parameters for non-formatted I/O functions are always byte counts, regardless of the current data width of the interface.
<b>iprintf, iscanf</b>	All formatted I/O functions work with GPIO. When formatted I/O is used with 16-bit data widths, the formatting buffers reassemble the data as a stream of bytes. On Windows, 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 <b>iscanf</b> termination, an END value must be specified using <b>igpioctrl</b> . See <i>Chapter 11 - SICL Language Reference</i> .
<b>itermchr</b>	For 16-bit data widths, only low (least-significant) byte is used.
<b>ixtrig</b>	Provides a method of triggering using either the CTL0 or CTL1 control lines. This function pulses the specified control line for approximately 1 or 2 $\mu$ sec. The following constants are defined: <b>I_TRIG_STD</b> Pulse CTL0 line <b>I_TRIG_GPIO_CTL0</b> Pulse CTL0 line <b>I_TRIG_GPIO_CTL1</b> Pulse CTL1 line
<b>itrigger</b>	Same as <b>ixtrig</b> ( <b>I_TRIG_STD</b> ). Pulses the CTL0 control line.
<b>iclear</b>	Pulses the P_RESET line for at least 12 $\mu$ sec, aborts any pending writes, discards any data in the receive buffer, and resets any error conditions. Optionally, clears the Data Out port, depending on the configuration specified via <b>IO Config</b> .
<b>ionsrq</b>	Installs a service request handler for this session. The concept of service request (SRQ) originates from GPIB. On a GPIB 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.
<b>ireadstb</b>	Although <b>ireadstb</b> is for device sessions only, since GPIO has no device sessions, <b>ireadstb</b> 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  
Sessions Interrupts

There are specific interface session interrupts that can be used. See **isetintr** in *Chapter 11 - SICL Language Reference* for information on the interface session interrupts for GPIO.

Using SICL with GPIO  
Using GPIO Interface Sessions

## Example: GPIO Interface Session (C)

```
/* gpiomeas.c
This program:
- Creates a 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 */

    #if defined (__BORLANDC__) && !defined (__WIN32__)
    _InitEasyWin(); /* required for Borland EasyWin programs */
    #endif

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

    /* open GPIO interface session, with 3 sec timeout*/
    id = iopen("gpio");
    itimeout(id, 3000);

    /* 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);
    /* This call is a no-op for WIN32 applications.*/
    _siclcleanup();

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

```

## Example: GPIO Interface Session (Visual Basic)

```
` This program:
` - Creates a GPIO session with timeout and error checking
` - Signals the device with a CTL0 pulse
` - Reads the device's response using formatted I/O
`
Sub cmdMeas_Click ()
    Dim id As Integer           ` device session id
    Dim retval As Integer       ` function return value
    Dim buf As String           ` buffer for displaying
    Dim real_data As Double     ` data from device

    ` Set up an error handler within this subroutine that will
    ` be called if a SICL error occurs.
    On Error GoTo ErrorHandler

    ` Disable the button used to initiate I/O while I/O is
    ` being performed.
    cmdMeas.Enabled = False

    ` Open an interface session using a known symbolic name
    id = iopen("gpio12")

    ` Set the I/O timeout value for this session to 3 sec
    Call itimeout(id, 3000)

    ` Setup formatted I/O configuration
    Call igpiosetWidth(id, 8)
    Call igpioctrl(id, I_GPIO_READ_EOI, 10)

    ` Signal the device to take a measurement
    Call itrigger(id)

    ` Get the data
    retval = ivscanf(id, "%lf%t", real_data)

    ` Display the response as string in a Message Box
    buf = Str$(real_data)
    retval = MsgBox(buf, MB_OK, "GPIO Data")

    ` Close the device session.
    Call iclose(id)
```

## Using SICL with GPIO

### Using GPIO Interface Sessions

```
` Enable the button used to initiate I/O
cmdMeas.Enabled = True

Exit Sub

ErrorHandler:
` Display the error message string in a Message Box
retval = MsgBox(Error$, MB_ICONEXCLAMATION, "SICL Error")

` Close the device session if iopen was successful.
If id <> 0 Then
    iclose (id)
End If

` Enable the button used to initiate I/O
cmdMeas.Enabled = True
Exit Sub

End Sub

` The following routine is called when the application's
` Start Up form is unloaded. It calls siclcleanup to
` release resources allocated by SICL for this
` application. `

Sub Form_Unload (Cancel As Integer)
    Call siclcleanup ` Tell SICL to clean up for this task
End Sub
```

### Example: GPIO Interrupts

```
/* gpointr.c
   This program:
   - Creates a GPIO session with error checking
   - Installs an interrupt handler and enables EIR interrupts
   - Waits for EIR; invokes the handler for each interrupt
*/

#include <sicl.h>

void SICLCALLBACK handler(id, reason, sec)
INST id;
int reason, sec;
{
```

## Using SICL with GPIO

### Using GPIO Interface Sessions

```
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 */

    #if defined (__BORLANDC__) && !defined (__WIN32__)
    _InitEasyWin(); /* required for Borland EasyWin programs */
    #endif

    /* 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) {
        iwaitdhr(0); /* optional timeout can be specified here*/
    }
}
```

## Using SICL with GPIO

### Using GPIO Interface Sessions

```
/* iwaitdhr performs an automatic iintron(). If your program
 * does concurrent processing, instead of waiting 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(id);

/* This call is a no-op for WIN32 applications. */
_siclcleanup();
}
```