CONTROLLING ICS's 8065 WITH AGILENT's SICL LIBRARY

INTRODUCTION

ICS's new 8065 Ethernet-to-GPIB Controller is a VXI-11 compatible device. The 8065 responds to all VXI-11.2 interface commands to operate as an IEEE-488.1 GPIB Controller and control the GPIB interface. This includes transfer data to/from a device, pulse the IFC line, send Device Clear and Device Triggers, set/reset REN, set/reset ATN, perform Serial Polls and to read back the status of the REN, NDAC and SRQ lines. The 8065 also responds to the VXI-11.3 device commands as a Ethernet-to-GPIB gateway to communicate with a GPIB instrument. Note that VXI-11.3 commands only let the 8065 communicate with a GPIB device and do not include the GPIB Controller functions.

VISA layers or libraries were created in the early 1990s to give the Test and Measurement community a standard Application Interface that could be called by anyone's test program. The VISA layers connected with the manufacturer's existing drivers and also provided a VXI-11 client output. VISA libraries were initially used with graphical programs like Agilent's VEE and National Instruments' LabView. Later other programmers begin writing C and Visual Basic language programs with VISA calls.

The problem is most VISA libraries handle VXI-11.3 commands with only minimal or no VXI-11.2 support. This limits the 8065's use as a GPIB Controller since without the VXI-11.2 interface commands, you can not implement IEEE-488.2 protocols like FindLstn nor do simple IEEE-488.1 functions such as writing GPIB Command strings.

AGILENT'S VISA/SICL LIBRARY

Hewlet-Packard (now Agilent) had written their SICL library before the VISA Specification was generated. Their SICL Library was essentially the same concept as the VISA Specification except it did not have the VISA standard API. However it has a complete VXI-11.2 and VXI-11.3 capability. Agilent's VISA layer sits on top of their SICL library and calls the SICL library to interface with the hardware drivers and to be the VXI-11 client. Unfortunately Agilent's VISA layer appears to only make VXI-11.3 calls which limits its usefulness with the 8065.

THE SOLUTION

The solution is to write programs for the 8065 that make SICL calls and to bypass Agilent's VISA layer altogether. SICL's VXI-11.2 commands enable all of the 8065's GPIB Controller capabilities and give the programmer a complete IEEE-488.1 and 488.2 capability. At the same time, direct SICL calls speed up the program and eliminate a useless software layer.

This Application Note covers how to write a SICL program in Visual Basic and includes a complete example program (SICL_kybd) that you can use as the starting point for your own program. SICL_kybd is an interactive keyboard control program that lets you control GPIB instruments directly from your keyboard. The example is complete with the source files, comments and a help file.

C and other language programmers can follow the steps outlined in this Application Note to create their own SICL program.

SICL LIBRARY AVAILABILITY

Agilent has upgraded their VISA/SICL libraries with their latest release of the Agilent IO Libraries Suite 14.1. The IO Libraries Suite can be downloaded from Agilent's website and is currently licensed at no charge to any user of their equipment. (If you don't have an HP/Agilent instrument in your company, you probably shouldn't be reading this application note.)

Agilent supplies an excellent guide to SICL programming (SICL User's Guide) that lists all of SICL's functions and provides easy to follow instructions for creating a LAN Session and for controlling GPIB devices through a LAN Interface like the 8065. The on-line Help file defines all of the SICL functions.

If you have done any amount of GPIB programming, the SICL library is pretty easy to use. The SICL keyboard example includes functions for completeness that the average programmer might never implement and uses just 15 SICL functions. These are the SICL functions for GPIB Device Sessions, GPIB Interface Sessions and some SICL LAN Functions.
PROGRAM EXAMPLE

ICS has written an interactive SICL keyboard program to show how to use Agilent's SICL Library. Figure 1 shows the SICL_kybd program's main form. The program listing is shown in Figure 2 at the end of this Application Note.

SICL_kybd is a fairly simple program that creates links to the 8065 and to a device so that the 8065 can transfer data to/from the device and send it other GPIB commands. VB Control buttons are provided to initiate the functions. They are enabled as the user steps through the program to give the user a feel for what he/she can do next. Two timer routines are included as part of the program.

Timer1 runs the IDN and the CMD loops which the user can turn on or off. The IDN loop continuously queries a device's IDN message until stopped. The IDN message is placed in the txtResults window. The CMD Loop executes the command in the comboCmd box. If the command string contains a '?', the read function is called to query the device. The response is placed in the txtResults window. A loop counter is incremented each time Timer1 executes the command.

Timer2 runs the background KeepAlive function. Refer to the Keep Alive comments on page 3 and in the 8065 Instruction Manual.

The program is not as complicated as it looks like at first glance. The functions include many VB Control commands that enable/disable the controls to guide the user through the program. Since SICL_kybd was adapted from an existing GPIB program it also has many variables for GPIB commands. Ignore them and just look for the SICL-GPIB control commands. The complete SICL_kybd program can be downloaded from ICS's website.

The remainder of this application note describes how the program was put together.

LAN PROGRAMMING BASICS

VXI-11 compliant products with LAN interfaces like the 8065 need to be identified and linked to before you can use the 8065 to control the GPIB bus. The SICL_kybd program has a comboBox where the user can enter the 8065's IP address. The Create Link button calls the cmdLink function that closes any open links and then opens an interface link to the 8065's IP address. The correct interface link format for the 8065 is:

\[
\text{intfc = iopen(“lan:vxi-11[192.168.0.254]:gpib“)}
\]

Note that the command ends with just 'gpib' specified for the interface link to the 8065. The IP address shown above is for the 8065's default IP address. The program actually inserts the IP address from the comboBox if the user entered an address in the comboBox.

At this point you can send the 8065 commands to pulse the ICS line, to set/reset the ATN and REN lines, and to read back some GPIB bus signals.

INSTRUMENT COMMUNICATION

VXI-11 instrument communication requires that you identify and link to the instrument that you want to control. The SICL_kybd program has a Find Instruments button that calls cmdFindInst. cmdFindInst is a simple 488.2 Find Listener protocol routine that only checks for devices with primary GPIB addresses. The results are shown in the comboInst box next to the Find Instrument button. The user selects the desired instrument address and clicks the Create Link button to link to the instrument.

The cmdFindInst routine checks all GPIB addresses from 0 to 30 for a low NRFD line from an addressed GPIB device. It skips the 8065's GPIB address and it does not check for secondary addresses. The user can add secondary addresses to the test to create a true general purpose Find Listener routine. The routine creates an address list, addrlist(I), which can be saved and used in other 488.2 protocols such as FindRQS.

The Create (Instrument) Link button calls the cmdLinki function which closes any open device link and then opens a link to the specified device. The SICL_kybd program was designed to control only one device at a time. However in a real application, the user would create links to multiple devices at the same time in his program. The correct device link format is:

\[
\text{dev = iopen(“lan:vxi-11[192.169.0.254]:gpib,”) + Str$(Device)}
\]

The IP address is the same address used for the 8065 link. Note that the device link command ends with 'gpib,n' to specify a GPIB device with primary address of \( n \).

At this point all of the device GPIB functions like write/read, Device Trigger, Device Clear and Serial Poll can be exercised. If the Device Trigger or Device Clear commands are called with the device's handle, dev, the device is addressed to listen and the GPIB SDC or GET command is sent to the specified device.
LIMITED NUMBER OF FUNCTIONS

Table 1 lists the 15 SICL functions used in the SICL_kybd program. Most GPIB programs use a less than 15 functions to communicate with and control GPIB instruments. A SICL program adds four additional functions to open/close links and to lock/unlock links.

Most of SICL_kybd’s functions include an ‘On Error GoTo ErrorHandler’ line before calling the SICL function and an Error handler routine at the end of the function. The SICL library returns errors as a Visual Basic error. For more details refer to Agilent’s SICL User Guide.

INSTRUMENT LOCKING

Instrument locking is important if you are using the 8065 in a situation where it can be accessed by multiple client applications. Without locking you would have no assurance that another client could not access one of your devices and change its settings and alter your test results. Locking is not necessary in the engineering lab but it should be incorporated into the program before it is released to production.

Locking is best done at the device level. By locking the devices, you prevent a second user from accessing them until you release the locks. See the section on locking in the 8065 manual.

The SICL_kybd program has an Auto Lock feature that can be used to automatically lock an instrument when a command or query is sent to an instrument and then unlock it after the command or when 8065 has received the response to the query.

CLOSING LINKS AND THE KEEP ALIVE FUNCTION

All LAN links need to be closed when you exit the program. Keep track of any open links and close all of them in the Exit routine. See the cmdExit routine.

The 8065 and your client may close the links when they discover that they have been inactive for a period of time. If your application has to pause for a period of time, such as over lunch breaks, equipment setup changes or UUT changes, it is best to implement a background KeepAlive function that will prevent link closure. The SICL_kybd program does this in the Timer2 function by momentarily opening and closing a second link to the device and by checking the GPIB bus status on the 8065 interface link. This is done on a once a minute basis when the main portion of the program is idle.

PROGRAM DEBUGGING

ICS provides an ErrorLog Utility program that retrieves error messages from the 8065 and displays them in a DOS box. The ErrorLog Utility is useful to run during program development time since it helps correct command syntax and other errors.

Note that SICL’s iopen function has two minor VXI-11 errors that are displayed by the ErrorLog Utility. They are a REN call to an instrument and a bad docomand. They cause the 8065 to blink its ERR LED and record the errors but do not stop the 8065 from opening the interface link. Agilent has prepared a patch to correct this problem.

SUMMARY

This Application Note has shown how to overcome the lack of VXI-11.2 commands in the popular VISA libraries by programming ICS’s 8065 Ethernet-to-GPIB Controller with calls to Agilent's SICL library. LabView and LabWindows users can install SICL by installing the Agilent VISA as the secondary VISA. This leaves the NI VISA in place and yet provides a way to make VXI-11.2 calls through the SICL library.

This Application Note also described how to write a Visual Basic program by examining ICS’s SICL_kybd program. SICL_kybd is an interactive instrument control program that was adapted from ICS’s VXI-11kybd program. Because it is a general purpose program it includes many flags and variables that are not necessary in a user’s test program. It also has extensive error checking to alert the user to any problem he may encounter. Notes and programming instructions are included to help the programmer avoid time consuming pitfalls.

SICL turned out to be a very easy library to use. Agilent’s documentation is very clear and easy to follow. The hardest part of converting a GPIB program to one that uses SICL calls was getting the address format correct for the iopen function.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iclear</td>
<td>Performs a GPIB interface clear (pulses IFC), which resets the GPIB interfaces or clears a device.</td>
</tr>
<tr>
<td>iclose</td>
<td>Closes a SICL session</td>
</tr>
<tr>
<td>igpibatnctl</td>
<td>Sets or clears the ATN line.</td>
</tr>
<tr>
<td>igpibbusstatus</td>
<td>Returns requested GPIB bus data.</td>
</tr>
<tr>
<td>igpibrenctl</td>
<td>Sets or clears the REN line.</td>
</tr>
<tr>
<td>igpibsendcmd</td>
<td>Sends data with ATN line set.</td>
</tr>
<tr>
<td>ilock</td>
<td>Locks a session to ensure exclusive use of a resource.</td>
</tr>
<tr>
<td>iopen</td>
<td>Opens a SICL session.</td>
</tr>
<tr>
<td>iread</td>
<td>Reads the data from a specified device or interface.</td>
</tr>
<tr>
<td>ireadstb</td>
<td>Reads the status byte from a specified device (Performs a Serial Poll)</td>
</tr>
<tr>
<td>itermchr</td>
<td>Defines a termination character condition.</td>
</tr>
<tr>
<td>itimeout</td>
<td>Sets GPIB bus timeout.</td>
</tr>
<tr>
<td>itrigger</td>
<td>Sends a GPIB trigger command (GET) to a specified device or interface.</td>
</tr>
<tr>
<td>iunlock</td>
<td>Unlocks a session to free the resource.</td>
</tr>
<tr>
<td>iwrite</td>
<td>Sends the data to a specified device or interface.</td>
</tr>
</tbody>
</table>
If mode& = 1 Then
    txtResults.Text = “ATN Asserted”
Else
    txtResults.Text = “ATN deasserted”
End If
End If
GoTo atnexit:
ErrorHandler:
    txtError.Text = “Check ATN error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
atnexit:
End Sub

Private Sub ckAutoLock_Click()
    If ckAutoLock.value = 1 Then
        If Sendlock = 1 Then                    'test for existing lock
            ckAutoLock.value = 0
        Else
            cmdLock.Enabled = False
            cmdUnlock.Enabled = False
        End If
    Else
        cmdLock.Enabled = True
        cmdUnlock.Enabled = True
    End If
End Sub

Private Sub ckCR_Click()
    On Error GoTo ErrorHandler
    If ckCR.value = 1 Then
        term% = 13                                   'set termination for CR or EOI
        Call itermchr(dev, term%)
    End If
    GoTo ckCRexit:
ErrorHandler:
    txtError.Text = “Termination character error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
ckCRexit:
End Sub

Private Sub ckLF_Click()
    On Error GoTo ErrorHandler
    If ckLF.value = 1 Then
        term% = 10                                   'set termination for LF or EOI
        Call itermchr(dev, term%)
    End If
    GoTo ckLFexit:
ErrorHandler:
    txtError.Text = “Termination character error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
ckLFexit:
End Sub

Private Sub cmdDT_Click()
    txtError.Visible = False
    combCmd.Text = “”
    txtResults.Text = “”
    If dev = 0 Then
        txtResults.Text = “No devices linked. Select and link to a de-
    End If
End Sub

Private Sub ckATN_Click()
    txtError.Visible = False
    If intfc <> 0 Then
        If ckATN.value = 1 Then
            mode& = 1
        Else
            mode& = 0
        End If
        On Error GoTo ErrorHandler
        Call igpibatnctl(intfc, mode&)                                  'set/clear
    End If
End Sub

Private Sub cmdCT_Click()
    txtError.Visible = False
    combCmd.Text = “”
    txtResults.Text = “”
    If dev = 0 Then
        txtResults.Text = “No devices linked. Select and link to a de-
    End If
End Sub

Figure 1    SICL Keyboard Program Listing
If id <> Ctlraddr% Then
    CmdStr$ = Chr$(63) + Chr$(64 + Ctlraddr%) + Chr$(32 + id)
    Call igpibsendcmd(intfc, CmdStr$, 3)
    Call igpibatnctl(intfc, 0)
    Call igpibbusstatus(intfc, 3, result%) 'get NDAC status
    If result% <> 0 Then
        addrlist%(i) = id
        i = i + 1
    End If
    CmdStr$ = Chr$(63)
    Call igpibsendcmd(intfc, CmdStr$, 1)
End If
Next id
addrlist%(i) = NOADDR 'NOADDR = GPIB address of -1
For i = 0 To 63                          'put found device addr in combbox
    If addrlist%(i) = NOADDR Then Exit For
    combInst.AddItem addrlist%(i), (i)
Next i
If i = 0 Then
    txtResults.Text = "FindInst: No VXI-11 devices found"
    GoTo finddexit:
ElseIf i = 1 Then
    txtResults.Text = "FindInst: Found " & Str$(i) & " VXI-11 device"
Else
    txtResults.Text = "FindInst: Found " & Str$(i) & " VXI-11 devices"
End If
txtResults.Text = txtResults.Text & NL & "Select a device and press Create Link"
msgSentFlg = 1 'sets flag to show that the link was exercised
GoTo finddexit:
ErrorHandler:
    txtError.Text = "Create device link error" & NL & Error$ & " - Retry"
    txtError.Visible = True
    Beep
    DTexit:
    End Sub

Private Sub cmdFindInst_Click()  
    txtError.Visible = False  
    cmdLinki.Enabled = False  
    cmdSend.Enabled = False  
    cmdRead.Enabled = False  
    cmdIDNtst.Enabled = False  
    cmdIDNtstoff.Enabled = False  
    cmdCmdtst.Enabled = False  
    cmdCmdtstoff.Enabled = False  
    cmdLock.Enabled = False  
    cmdUnlock.Enabled = False  
    cmdSDC.Enabled = False  
    cmdDT.Enabled = False  
    cmdSpoll.Enabled = False  
    ckCR.Enabled = False  
    ckLF.Enabled = False  
    ckAutoQuery.Enabled = False  
    ckAutoLock.Enabled = False  
    ckAutoLock.value = 0  
    ckByteCnt.Enabled = False  
    combInst.Clear  
    combCmd.Text = ""  
    NOADDR = -1  
    If dev <> 0 Then 'disable any prior device link
        Call iclose(dev)
        lbliLock.Visible = False
        dev = 0
    End If
    Call igpibbusstatus(intfc, 8, result%) 'get intfc GPIB addr
    Ctlraddr% = result%
    Dim addrlist%(32) 'get intfc GPIB addr
    limit% = 32
    foundaddr& = 0
    txtResults.Text = "FindInst: Wait while looking for VXI-11 devices"
    txtResults.Refresh
    On Error GoTo ErrorHandler  'install error handler
    i = 0
    For id = 0 To 30 'all GPIB primary addr
        If id <> Ctlraddr% Then
            CmdStr$ = Chr$(63) + Chr$(64 + Ctlraddr%) + Chr$(32 + id)
            Call igpibsendcmd(intfc, CmdStr$, 3)
            Call igpibatnctl(intfc, 0)
            Call igpibbusstatus(intfc, 3, result%) 'get NDAC status
            If result% <> 0 Then
                addrlist%(i) = id
                i = i + 1
            End If
            CmdStr$ = Chr$(63)
            Call igpibsendcmd(intfc, CmdStr$, 1)
        End If
    Next id
    addrlist%(i) = NOADDR 'NOADDR = GPIB address of -1
    For i = 0 To 63 'put found device addr in combbox
        If addrlist%(i) = NOADDR Then Exit For
        combInst.AddItem addrlist%(i), (i)
    Next i
    If i = 0 Then
        txtResults.Text = "FindInst: No VXI-11 devices found"
        GoTo finddexit:
    ElseIf i = 1 Then
        txtResults.Text = "FindInst: Found " & Str$(i) & " VXI-11 device"
    Else
        txtResults.Text = "FindInst: Found " & Str$(i) & " VXI-11 devices"
    End If
    txtResults.Text = txtResults.Text & NL & "Select a device and press Create Link"
    cmdLinki.Enabled = True
    MsgBoxFlg = True
    GoTo finddexit:
    ErrorHandler:
        txtError.Text = "Create device link error" & NL & Error$ & " - Retry"
        txtError.Visible = True
        Beep
        DTexit:
        End Sub

Private Sub cmdLink_Click()  'create Server link
    txtError.Visible = False
    txtError.Refresh
    cmdFindInst.Enabled = False
    cmdLinki.Enabled = False
    cmdSend.Enabled = False
    cmdRead.Enabled = False
    cmdIDNtst.Enabled = False
    cmdIDNtstoff.Enabled = False
    cmdCmdtst.Enabled = False
    cmdCmdtstoff.Enabled = False
    cmdLock.Enabled = False
    cmdUnlock.Enabled = False
    cmdSDC.Enabled = False
    cmdDT.Enabled = False
    cmdSpoll.Enabled = False
    ckCR.value = 0
    ckLF.value = 1
    ckAutoQuery.Enabled = False
    ckAutoLock.Enabled = False
    ckAutoLock.value = 0
    ckByteCnt.Enabled = False
    combInst.Clear
    combCmd.Text = ""
ckCR.Enabled = False  
ckLF.Enabled = False  
ckRen.Enabled = False  
ckATN.Enabled = False  
ckAutoLock.Enabled = False  
ckAutoLock.value = 0  
ckCR.value = 0  
ckLF.value = 1  
ckAutoQuery.Enabled = False  
ckByteCnt.Enabled = False  
combInst.Clear  
combCmd.Text = ""  
If dev <> 0 Then 'disable any prior device link  
   Call iclose(dev)  
   lbliLock.Visible = False  
   dev = 0  
End If  
If intfc <> 0 Then 'disable any prior link  
   Call iclose(intfc)  
   intfc = 0  
End If  
On Error GoTo ErrorHandler 'install error handler  
ip$ = combSrvrs.Text  
If ip$ = "" Then ip$ = "192.168.0.254"  
CmdStr$ = "lan;vxi-11[" + ip$ + "]:gpib" 'lan[128.10.0.3]:gpib Correct)  
intfc = iopen(CmdStr$) 'intfc refers to the 8065  
cmdFindInst.Enabled = True  
cmdIFC.Enabled = True  
cmdStatus.Enabled = True  
ckRen.Enabled = True  
ckATN.Enabled = True  
ckRen.value = 1  
ckATN.value = 1  
combSrvrs.Text = ip$  
server = ip$  
txtResults.Text = "Link created to server " & server  
optTimeout(0).Enabled = True  
optTimeout(1).Enabled = True  
optTimeout(2).Enabled = True  
optTimeout(3).Enabled = True  
optTimeout(1).value = True  
MsgSentFlg = 1 'sets flag to show that the link was exercised  
GoTo Linkiexit:  
ErrorHandler: 'install error handler  
ip$ = combSrvrs.Text  
If ip$ = "" Then ip$ = "192.168.0.254"  
CmdStr$ = "lan;vxi-11[" + ip$ + "]:gpib," + Str$(Device) 'lan[128.10.0.3]:gpib,device Correct)  
dev = iopen(CmdStr$) 'dev refers to the GPIB device  
txtResults.Text = "Link created to instrument at " & server & "," & Str$(pad%) & "," & Str$(sad%)  
cmdSend.Enabled = True  
cmdRead.Enabled = True  
cmdIDNtst.Enabled = True  
cmdIDNtstoff.Enabled = False  
cmdCmdtst.Enabled = True  
cmdCmdtstoff.Enabled = False  
cmdLock.Enabled = True  
cmdUnlock.Enabled = True  
cmdSDC.Enabled = True  
cmdDT.Enabled = True  
cmdSpoll.Enabled = True  
ckCR.Enabled = True  
ckLF.Enabled = True  
ckAutoLock.Enabled = True  
ckCR.value = 0  
ckLF.value = 1  
ckAutoQuery.Enabled = True  
ckByteCnt.Enabled = True  
If optTimeout(0).value = True Then Call optTimeout_Click(0)  
If optTimeout(1).value = True Then Call optTimeout_Click(1)  
If optTimeout(2).value = True Then Call optTimeout_Click(2)  
MsgSentFlg = 1 'sets flag to show that the link was exercised  
GoTo Linkiexit:  
Figure 1  SICL Keyboard Program Listing Continued
ErrorHandler:
    txtError.Text = “Create device link error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
    Lockexit:
End Sub

Private Sub cmdLock_Click()
On Error GoTo ErrorHandler
Call ilock(dev)
    txtError.Visible = False
    txtResults.Text = “Device “ & Str$(pad%) & “ locked”
    lbliLock.Visible = True
    lbliLock.Refresh
    Sendlock = 1  ’sets locked flag
    MsgSentFlg = 1  ’sets flag to show that the link was exercised
GoTo Lockexit:
ErrorHandler:
    txtError.Text = “Lock error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
Lockexit:
End Sub

Private Sub cmdUnlock_Click()
    On Error GoTo ErrorHandler
    Call iunlock(dev)
    txtError.Visible = False
    txtResults.Text = “Device “ & Str$(pad%) & “ unlocked”
    lbliLock.Visible = False
    Sendlock = 0  ’clears locked flag
    MsgSentFlg = 1  ’sets flag to show that the link was exercised
GoTo Unlockexit:
ErrorHandler:
    txtError.Text = “Unlock error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
Unlockexit:
End Sub

Private Sub cmdSDC_Click()
    txtError.Visible = False
    combCmd.Text = “
    txtResults.Text = “
    If dev = 0 Then
        txtResults.Text = “No devices linked. Select and link to a device”
        txtError.Visible = True
        Beep
    GoTo SDCexit:
    End If
    On Error GoTo ErrorHandler
    Call iclear(dev)
    txtError.Visible = False
    txtResults.Text = “Device Clear sent to device “ & Str$(pad%)  ’sets flag to show that the link was exercised
    MsgBox = 1  ’sets flag to show that the link was exercized
GoTo SDCexit:
ErrorHandler:
    txtError.Text = “Check REN error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
SDCexit:
End Sub

Private Sub cmdStatus_Click()
    txtError.Visible = False
    On Error GoTo ErrorHandler
    Call igpibbusstatus(intfc, 1, result%)  ’get REN
    If result% <> 0 Then result% = 1  ’corrects SICL 256 response to 1
    txtResults.Text = “REN =” & Str$(result%)
    Call igpibbusstatus(intfc, 2, result%)  ’get SRQ
    If result% <> 0 Then result% = 1
    txtResults.Text = txtResults.Text & NL & “SRQ =” & Str$(result%)
    Call igpibbusstatus(intfc, 3, result%)  ’get NDAC
    If result% <> 0 Then result% = 1
    txtResults.Text = txtResults.Text & NL & “NDAC =” & Str$(result%)
    GoTo queryexit:
ErrorHandler:
    txtError.Text = “Read Status error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
queryexit:
End Sub

Private Sub cmdSDC_Click()
    txtError.Visible = False
    combCmd.Text = “
    txtResults.Text = “
    If dev = 0 Then
        txtResults.Text = “No devices linked. Select and link to a device”
        txtError.Visible = True
        Beep
    GoTo SDCexit:
    End If
    On Error GoTo ErrorHandler
    Call iclear(dev)
    txtError.Visible = False
    txtResults.Text = “Device Clear sent to device “ & Str$(pad%)  ’sets flag to show that the link was exercised
    MsgBox = 1  ’sets flag to show that the link was exercized
GoTo SDCexit:
ErrorHandler:
    txtError.Text = “Check REN error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
SDCexit:
End Sub

Private Sub cmdStatus_Click()
    txtError.Visible = False
    On Error GoTo ErrorHandler
    Call igpibbusstatus(intfc, 1, result%)  ’get REN
    If result% <> 0 Then result% = 1  ’corrects SICL 256 response to 1
    txtResults.Text = “REN =” & Str$(result%)
    Call igpibbusstatus(intfc, 2, result%)  ’get SRQ
    If result% <> 0 Then result% = 1
    txtResults.Text = txtResults.Text & NL & “SRQ =” & Str$(result%)
    Call igpibbusstatus(intfc, 3, result%)  ’get NDAC
    If result% <> 0 Then result% = 1
    txtResults.Text = txtResults.Text & NL & “NDAC =” & Str$(result%)
    GoTo queryexit:
ErrorHandler:
    txtError.Text = “Read Status error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
queryexit:
End Sub

Private Sub cmdUnlock_Click()
    On Error GoTo ErrorHandler
    Call iunlock(dev)
    txtError.Visible = False
    txtResults.Text = “Device “ & Str$(pad%) & “ unlocked”
    lbliLock.Visible = False
    Sendlock = 0  ’clears locked flag
    MsgSentFlg = 1  ’sets flag to show that the link was exercised
GoTo Unlockexit:
ErrorHandler:
    txtError.Text = “Unlock error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
Unlockexit:
End Sub

Private Sub cmdSDC_Click()
    txtError.Visible = False
    combCmd.Text = “
    txtResults.Text = “
    If dev = 0 Then
        txtResults.Text = “No devices linked. Select and link to a device”
        txtError.Visible = True
        Beep
    GoTo SDCexit:
    End If
    On Error GoTo ErrorHandler
    Call iclear(dev)
    txtError.Visible = False
    txtResults.Text = “Device Clear sent to device “ & Str$(pad%)  ’sets flag to show that the link was exercised
    MsgBox = 1  ’sets flag to show that the link was exercized
GoTo SDCexit:
ErrorHandler:
    txtError.Text = “Check REN error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
SDCexit:
End Sub

Private Sub cmdStatus_Click()
    txtError.Visible = False
    On Error GoTo ErrorHandler
    Call igpibbusstatus(intfc, 1, result%)  ’get REN
    If result% <> 0 Then result% = 1  ’corrects SICL 256 response to 1
    txtResults.Text = “REN =” & Str$(result%)
    Call igpibbusstatus(intfc, 2, result%)  ’get SRQ
    If result% <> 0 Then result% = 1
    txtResults.Text = txtResults.Text & NL & “SRQ =” & Str$(result%)
    Call igpibbusstatus(intfc, 3, result%)  ’get NDAC
    If result% <> 0 Then result% = 1
    txtResults.Text = txtResults.Text & NL & “NDAC =” & Str$(result%)
    GoTo queryexit:
ErrorHandler:
    txtError.Text = “Read Status error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
queryexit:
End Sub

Private Sub cmdUnlock_Click()
    On Error GoTo ErrorHandler
    Call iunlock(dev)
    txtError.Visible = False
    txtResults.Text = “Device “ & Str$(pad%) & “ unlocked”
    lbliLock.Visible = False
    Sendlock = 0  ’clears locked flag
    MsgSentFlg = 1  ’sets flag to show that the link was exercised
GoTo Unlockexit:
ErrorHandler:
    txtError.Text = “Unlock error” & NL & Error$ & “ - Retry”
    txtError.Visible = True
    Beep
Unlockexit:
End Sub
GoTo renexit:
ErrorHandler:
txtError.Text = “Check REN error” & NL & Error$ & “ - Retry”
txtError.Visible = True
Beep
renexit:
End Sub
Private Sub cmdCmdtst_Click()
If combCmd.Text = “” Then
    Beep
    txtResults.Text = “No command specified. Enter a command before starting the Command Loop”
Else
    CMDloopFlg = 1
    TestHalt = 0
    Timer1.Interval = 250
    cmdCmdtst.Enabled = False
    cmdCmdtstoff.Enabled = True
End If
End Sub
Private Sub cmdCMDtstoff_Click()
    CMDloopFlg = 0
    cmdCmdtst.Enabled = True
    cmdCmdtstoff.Enabled = False
    If (CMDloopFlg = 0) And (IDNTestFlg = 0) Then
        txtCount.Visible = False
        lblCount.Visible = False
        LpCount = 0
        TestHalt = 0
        Timer1.Interval = 0
    End If
End Sub
Private Sub cmdExit_Click()
If dev <> 0 Then                          ‘disable any prior device link
    Call iclose(dev)
    lbliLock.Visible = False
    dev = 0
End If
If intfc <> 0 Then                         ‘disable any prior interface link
    Call iclose(intfc)
End If
End
End Sub
Private Sub cmdHelp_Click()
’ HTML Help file launched in response to a button click:
’ Private Sub HH_DISPLAY_Click()
’ hWnd is a Long defined elsewhere to be the window handle
’ that will be the parent to the help window.
Dim hWndHelp As Long
    ‘The return value is the window handle of the created help window.
    hWndHelp = HtmlHelp(hWnd, “SICLkybd_help.chm”, HH_DISPLAY_TOPIC, 0)
    ‘ cdhHelp.HelpFileDialog = “\help\SICLkybd_help.chm”
’ cdhHelp.ShowHelp
End Sub
Private Sub cmdIDNtst_Click()
    combCmd.Text = “*idn?”
    Timer1.Interval = 250
    IDNTestFlg = 1
    TestHalt = 0
    cmdIDNtst.Enabled = False
    cmdIDNtstoff.Enabled = True
End Sub
Private Sub cmdIDNtstoff_Click()
    IDNTestFlg = 0
    cmdIDNtst.Enabled = True
    cmdIDNtstoff.Enabled = False
    If (CMDloopFlg = 0) And (IDNTestFlg = 0) Then
        txtCount.Visible = False
        lblCount.Visible = False
        LpCount = 0
        TestHalt = 0
        Timer1.Interval = 0
    End If
End Sub
Private Sub cmdIFC_Click()
    txtError.Visible = False
    combCmd.Text = “”
    On Error GoTo ErrorHandler
    Call iclear(intfc)                                   ‘pulse IFC
    ckRen.value = 1
    ckATN.value = 0
    IDNTestFlg = 0
    cmdIDNtst.Enabled = True
    cmdIDNtstoff.Enabled = False
    CMDloopFlg = 0
    cmdCmdtst.Enabled = False
    cmdCmdtstoff.Enabled = True
    txtCount.Visible = False
    lblCount.Visible = False
    txtResults.Text = “IFC Sent”
    GoTo IFCexit:
ErrorHandler:
txtError.Text = “Send IFC error” & NL & Error$ & “ - Retry”
txtError.Visible = True
Beep
IFCexit:
End Sub
Private Sub cmdIOloop_Click()
    IOloopFlg = 1
End End Sub
Private Sub cmdIOloopoff_Click()
    IOloopFlg = 0
    If (IOloopFlg = 0) And (IDNTestFlg = 0) Then
        txtCount.Visible = False
        lblCount.Visible = False
        LpCount = 0
        End If
    End If
End End Sub
Private Sub cmdRead_Click()
    txtError.Visible = False
    If dev = 0 Then
        txtResults.Text = “No devices linked. Select a device and create link”
If Outstring$ = "" Then
    txtResults.Text = "Device command box empty, nothing sent to the device"
    Beep
    GoTo Sendexit:
End If
If ckCR = 1 Then Outstring$ = Outstring$ & Chr$(13)
If ckLF = 1 Then Outstring$ = Outstring$ & Chr$(10)
bufsize& = Len(Outstring$)
endi% = 1                             'assert EOI
On Error GoTo ErrorHandler
Call iwrite(dev, Outstring$, bufsize&, endi%, outcount&)
msgSentFlg = 1       'sets flag to show that the link was exercised
If (ckAutoQuery.value = 1) And (InStr(Outstring$, "?") <> 0)
    Then
        Call cmdRead_Click
    Else
        txtResults.Text = "Send String => " + Outstring$
        If ckAutoLock.value = 1 Then Call cmdUnlock_Click
        Sendlock = 0                      'reset flag
        End If
End If
GoTo Sendexit:
ErrorHandler:
    txtError.Text = "Send error" & NL & Error$ & " - Retry"
    txtError.Visible = True
    Beep
Sendexit:
End Sub
Private Sub cmdSPoll_Click()
    txtError.Visible = False
    Outstring$ = combCmd.Text
    txtResults.Text = "Serial poll response => " + Str$(rdg%)
    MsgSentFlg = 1       'sets flag to show that the link was exercised
End If
On Error GoTo ErrorHandler
Call ireadstb(dev, rdg%)
msgSentFlg = 1       'sets flag to show that the link was exercised
If (ckAutoQuery.value = 1) And (InStr(Outstring$, "?") <> 0)
    Then
        Call cmdRead_Click
    Else
        txtResults.Text = "Send error" & NL & Error$ & " - Retry"
        txtError.Visible = True
        Beep
Sendexit:
End Sub
Private Sub Form_Load()
    Rev$ = "Revised 12-20-2005"
    cmdLink.Enabled = True
    cmdLinki.Enabled = False
    cmdFindInst.Enabled = False
    cmdSend.Enabled = False
    cmdRead.Enabled = False
End Sub
Figure 1     SICL Keyboard Program Listing Continued
cmdIDNtst.Enabled = False
cmdIDNtstoff.Enabled = False
cmdCmdtst.Enabled = False
cmdCmdtstoff.Enabled = False
cmdIFC.Enabled = False
cmdStatus.Enabled = False
cmdDT.Enabled = False
cmdSDC.Enabled = False
cmdSpoll.Enabled = False
cmdLock.Enabled = False
cmdUnlock.Enabled = False
cmdExit.Enabled = True
ckAutoLock.Enabled = False
ckATN.Enabled = False
ckRen.Enabled = False
ckCR.Enabled = False
ckLF.Enabled = False
cmAutoQuery.value = 1
ckByteCnt.value = 0
lbliLock.Visible = False
lblKeepAliveMsg.Visible = False
optTimeout(0).Enabled = False
optTimeout(1).Enabled = False
optTimeout(2).Enabled = False
optTimeout(3).Enabled = False
SPACE80S = Space$(100)  '100 spaces
NL = Chr(13) + Chr(10)
txtRev.Text = Rev$
txtError.Text = ""  'clear label and text box
txtResults.Text = ""
combCmd.Text = ""
TypeVar = "GPIB"  'default to GPIB control
OutFlag = 0
FoundFlg = 0
BD% = 0  'define initial values
dev = 0
intfc = 0
bddev% = 0
addr% = 4
Device = 4
eos% = 10
Testnum = 100
IDNTestFlg = 0
CMDloopFlg = 0
IOloopFlg = 0
txtnull.Visible = False
LpCount = 0
Timer1.Interval = 0
txtResults.Enabled = True
txtError.Enabled = False
ErrFlag = 0
IDNTestFlg = 0
IOloopFlg = 0
combCmd.AddItem "*esr?", 0
combCmd.AddItem "*idn?", 1
combCmd.AddItem "*stb?", 2
FirstTimeFlg = 0
App.HelpFile = App.Path & "\GPIBkybd2.chm"
End Sub

Private Sub optTimeout_Click(Index As Integer)
Select Case Index
Case 0
    Call itimeout(intfc, 1000)  'time in milliseconds
    If dev <> 0 Then Call itimeout(dev, 1000)
End Select
End Sub

Private Sub Timer1_Timer()
If (CMDloopFlg = 0) And (IDNTestFlg = 0) Then
    'txtCount.Visible = False
    'LpCount = 0
    GoTo Timerexit:
End If
If TestHalt = 1 Then GoTo Timerexit:
If IDNTestFlg = 1 Then
    If ckAutoLock.value = 1 Then
        Call cmdLock_Click
        If Er% <> 0 Then
            Beep
            IDNTestFlg = 0
            GoTo Timerexit:
        End If
        Sendlock = 1  'set Sendlock flag for read routine
    End If
    Outstring$ = "*IDN?"
    If ckCR = 1 Then Outstring$ = Outstring$ + Chr$(13)
    If ckLF = 1 Then Outstring$ = Outstring$ + Chr$(10)
    bufsize& = Len(Outstring$)
    on Error GoTo ErrorHandler
    txtError.Text = "Write error" & NL & Error$ & " - Retry"
    Call iwrite(dev, Outstring$, bufsize&, endi%, outcount&)
    txtError.Visible = False
    MsgSentFlg = 1  'sets flag to show that the link was exercised
    If (ckAutoQuery.value = 1) And (InStr(Outstring$, "?") <> 0) Then
        txtError.Text = "Read error" & NL & Error$ & " - Retry"
        Call cmdRead_Click
    End If
End If
If CMDloopFlg = 1 Then  'loop use the string in the combCmd.
    If combCmd.Text = "" Then
        Beep
        txtError.Text = "Command box empty"
        txtError.Visible = True
    Else
        If ckAutoLock.value = 1 Then
            Call cmdLock_Click
            If Er% <> 0 Then
                Beep
                CMDloopFlg = 0
                GoTo Timerexit:
            End If
        End If
        Sendlock = 1  'set Sendlock flag for read routine
    End If
End If
End If
End Sub

Figure 1  SICL Keyboard Program Listing Continued
If ckCR = 1 Then Outstring$ = Outstring$ + Chr$(13)
If ckLF = 1 Then Outstring$ = Outstring$ + Chr$(10)
bufsize& = Len(Outstring$)
endi% = 1                             'assert EOI
On Error GoTo ErrorHandler
Call iwritedev, Outstring$, bufsize&, endi%, outcount&
MsgSentFlg = 1                  'sets flag to show that the link was exercised
If (ckAutoQuery.value = 1) And (InStr(Outstring$, "?") <> 0) Then
  txtError.Text = "Read error" & NL & Error$ & " - Retry"
  Call cmdRead_Click
End If
End If
End If
Timerexit1:       txtCount.Visible = True
  lblCount.Visible = True
  LpCount = LpCount + 1
  txtCount.Text = LpCount
  GoTo Timerexit:
ErrorHandler:
  txtError.Visible = True
  Beep
  If ckAutoLock.value = 1 Then Call cmdUnlock_Click
  Sendlock = 0                    'reset flag
  IDNTestFlg = 0
  CMDloopFlg = 0
  Timerexit:
End Sub

Private Sub Timer2_Timer()      'runs 1 minute link keep-alive functions
  If MsgSentFlg = 0 And intfc <> 0 Then      '0 needs a keep alive exercise
    lblKeepAliveMsg.Visible = True
    lblKeepAliveMsg.Refresh
    If dev <> 0 Then
      CmdStr$ = "lan;vxi-11[192.168.0.254]:gpib," + Str$(Device)
      dev2 = iopen(CmdStr$)
      'lan[128.10.0.3]:gpib Correct)
      Call iclose(dev2)
      'txtResults.Text = "Keep Alive Timer exercised the link"
      Call igpibbusstatus(intfc, 1, result%)   'get REN
      Tdelay(0.1)
      lblKeepAliveMsg.Visible = False
      End If
      MsgSentFlg = 0              'clears flag - link exercised
      End If
  End Sub

Private Sub cmdSet.Enabled = True
End Sub
Private Sub combCmd_KeyPress(KeyAscii As Integer)
  If KeyAscii = (13) Then
    Call cmdSend_Click
  End If
End Sub

LabView and labWindows are trademarks of National Instruments, Austin, Texas
SICL is a trademark of Agilent Corporation, Palo Alto, California

ICS Electronics 7034 Commerce Circle, Pleasanton, CA 94588 Phone: (925) 416-1000 Fax: (925) 416-0105