IC S
ELECTRONIC S
division of Systems West Inc.

IEEE 488 APPLICATION BULLETIN

CONVERTING OLDER GPIB PROGRAMS FOR ICS's GPIB CONTROLLERS

INTRODUCTION

As computer systems age and need to be replaced, test engineers are faced with the task of upgrading the test system and at the same time trying to save the existing test programs. The cost of the test programs are often more than the cost of the hardware. Replacing the test programs is very expensive because the original test engineers are typically gone and new engineers are not familiar with the test requirements and with the other considerations that were taken into account in the original program. Also any changes in the test software may cause the complete test program to have to be re-validated by the end customer.

This application note deals with two ways to change or update the GPIB Controller with a minimum of changes to the test program.

GENERAL CONCEPT

Most test programs only use a few GPIB instructions - two or three instructions to initialize the GPIB Controller and then three or four other instructions that are repeated throughout the program. This makes the job of replacing the GPIB Controller instructions fairly easy.

ICS's GPIB Controllers use a command set that is compatible with National Instruments' 'ib' and the NI 488.2 command sets. This gives the user a wide choice of GPIB Controller types and means that the program will probably never have to be updated again. While this Application Note deals specifically with the conversion of a Rocky Mountain Basic Program to ICS's 488.2 Commands, the concept remains the same for updating test programs written in other languages.

GLOBAL REPLACEMENT

One way to change GPIB Bus Controllers is to do a global edit and replace the original commands with the equivalent ICS 488.2 commands for ICS's GPIB Bus Controllers. The 488.2 commands are recommended over the 'ib' commands because of their ease of use.

INITIALIZATION

Figure 1 shows a simple Rocky Mountain Basic program. The first four commands. IOABORT, IOCLEAR, IOTIMEOUT and IOEOI are used to initialize the GPIB Controller and have to be replaced to change over to an ICS GPIB Controller. While the initialization section often gets a major rewrite, it doesn't matter since it is typically a small part of the test program and does not affect the logic of the program.

The first step is to replace the original QBSETUP include file with ICS's GPIB-32.BAS and ICSVB.BAS files. If this program is being converted into Visual Basic, ICS's GPIB-32.BAS and ICSVB.BAS would included in the Project and the include statement is not used.

In the Initialization section SendIFC replaces IOABORT, ibSRE replaces IOCLEAR and ibTMO replaces IOTIMEOUT. ibSRE and ibTMO are used here since there are no equivalent 488.2 commands.

The three variables are also modified. ISC& is set to 0 for the first GPIB Controller. Instead of the 7 used in Rocky Mountain Basic. Also the 7 is stripped from the device's GPIB addresses. TIMEOUT! is set equal to T10s for 10 seconds.

The new initialization sequence becomes:

```
ISC% = 0
ANA%= 16
TIMEOUT = T10s
CALL SendIFC(ISC%)
   IF (ibsta% and EERR) Then Call gpiberr
CALL ibSRE(ISC%,1)
   IF (ibsta% and EERR) Then Call gpiberr
CALL ibTMO(ISC%, TIMEOUT)
   IF (ibsta% and EERR) Then Call gpiberr
CLS
```

```
REM $INCLUDE: 'C:\HPIB\QBSETUP'
 ISC& = 7
 ANA& = 716
 TIMEOUT! = 10
 CALL IOABORT(ISC&)
  IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR
 CALL IOCLEAR(ISC&)
  IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR
 CALL IOTIMEOUT(ISC&, TIMEOUT!)
  IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR
 CALL IOEOI(ISC&, 0)
  IF PCIB.ERR <> NOERR THEN ERROR PCIB.BASERR
 CLS

 CALL IOREMOTE(ANA&)
 CODE$ = "TALKLIST;"
 CALL IOOUTPUTS(ANA&, CODE$, LEN(CODE$))
 CODE$ = "IDN?"
 CALL IOOUTPUTS(ANA&, CODE$, LEN(CODE$))
 RCV$ = SPACE$(72)
 ACT% = 0
 MAX% = 72
 CALL IOENTERS(ANA&, RCV$, MAX%, ACT%)
 PRINT "ANA MODEL "; RCV$
 CODE$ = "OPC;PRES;"
 CALL IOOUTPUTS(ANA&, CODE$, LEN(CODE$))
 END
```

Figure 1 Original Rocky Mountain Basic Program

BODY OF PROGRAM

There are only three more GPIB Commands used in this sample program, IOREMOTE, IOOUTOUT and IOENTER. In a larger program there would be many occurrences of these instructions and possibly a couple of additional instructions like IOSPOLL

IOREMOTE is used to place device ANA% into the remote state. This command is really unnecessary as the next instruction will put the device into the remote state when it writes an instruction out to the device. IOREMOTE can be replaced with EnableRemote. EnableRemote uses a device address list and is coded as:

```
DIM addrlist%(2)
addrlist%(0) = ANA%
sddrlist%(1) = NOADDR
Call EnableRemote(ISC%, addrlist%())
```

IOOUTPUT is used to send the string CODE$ to device ANA%. IOOUTPUT is replaced by Send. The arguments for Send are Bd%, dev%, string$ and terminator%. Bd% is the same as ISC% defined above The terminator can be set to linefeed which is the common terminator for a HP test program. Each IOOUTPUT is replaced by:

```
Send(ISC%, ANA%, CODE$, TERM%)
```

IOENTER is used to read data from device ANA%. The equivalent 488.2 command is Receive. The arguments for Receive are Bd%, dev%, RCV$ and Terminator%. Again Bd$% is ISC% and equal to 0. Dev% is ANA%. Each IOENTER is replaced with:

```
Receive(ISC%, ANA%, RCV$, Term%)
```

The Global Replacement Method simply replaces these commands with the above substitutions. Figure 2 shows the example program converted by the Global replace concept.

```
REM $INCLUDE:  GPIB-32.BAS and ICSVB.BAS files.
ISC% = 0
ANA%= 16
TIMEOUT = T10s
CALL SendIFC(ISC%)
   IF (ibsta% and EERR) Then Call gpiberr
CALL ibSRE(ISC%,1)
   IF (ibsta% and EERR) Then Call gpiberr
CALL ibTMO(ISC%, TIMEOUT)
   IF (ibsta% and EERR) Then Call gpiberr
CLS

DIM addrlist%(2)
addrlist%(0) = ANA%
sddrlist%(1) = NOADDR
Call EnableRemote(ISC%, addrlist%())
CODE$ = "TALKLIST;"
Send(ISC%, ANA%, CODE$, TERM%)
CODE$ = "IDN?"
Send(ISC%, ANA%, CODE$, TERM%)
RCV$ = SPACE$(72)
ACT% = 0
MAX% = 72
Receive(ISC%, ANA%, RCV$, Term%)
PRINT "ANA MODEL "; RCV$
CODE$ = "OPC;PRES;"
Send(ISC%, ANA%, CODE$, TERM%)
END
```

Figure 2 Global Replacement Program Listing

SUBROUTINE METHOD

The Subroutine method differs from the Global Replacement Concept in that it uses the call to the existing GPIB commands as a call to new subroutines with ICS's equivalent GPIB commands. The subroutine makes any necessary parameter conversions, calls the equivalent ICS command and returns any necessary parameters.

The user has to decide how to handle the program initialization. Is it worth it to make subroutines of the IOABORT, IOCLEAR, IOTIMEOUT and IOEOI commands for a one time use or just replace the commands with a new initialization sequence tailored to ICS's GPIB Controllers? Probably not. So since the initialization commands are only called once we will use the same initialization sequence developed in the Global Replacement Method.

IOREMOTE, IOOUTPUT and IOENTER will be converted in to subroutines as they represent GPIB commands that would be repeated many times in a larger program.

IOREMOTE now becomes the name of a new subroutine with the arguments ANA% and CODE$. The subroutine creates addrlist and calls the REMOTE command.

```
IOREMOTE(ANA%)
   DIM addrlist%(2)
   addrlist%(0) = ANA%
   sddrlist%(1) = NOADDR
   Call EnableRemote(ISC%, addrlist%())
RETURN
```

IOOUTPUTS becomes the name of a new subroutine to send the CODE$ string to device ANA%. Send has four arguments: BD%, DEV%, String$ and TERM%. BD% becomes ISC% and DEV% becomes ANA%. TERM% is set to NLend% for a linefeed. The subroutine then calls Send. NLend% and ISC% are global variables.

```
IOOUTPUTS(ANA%, CODE$, LEN)
   Send(ISC%, ANA%, CODE$, NLend%)
RETURN
```

IOINPUTs becomes the name of a new subroutine to read a string from device ANA%. Receive has four arguments: BD%, DEV%, InString$ and TERMINATION%. BD% becomes ISC% and DEV% becomes ANA%. TERMIMATION% is set to linefeed. The subroutine then calls Receive. ISC% is a global variable.

```
IOENTERS(ANA%, RCV$, MAX%, ACT%)
   Term% = 10
   Receive(ISC%, ANA%, RCV$, Term%)
RETURN
```

Figure 3 shows how the final program looks.

SUMMARY

This Application Note has shown two ways a test engineer can update an older test program to use a modern GPIB Controller without rewriting the program or without disturbing the program's flow. The Subroutine Method is the preferred method since it involves less changes to the original program.

```
REM $INCLUDE:  GPIB-32.BAS and ICSVB.BAS files.
ISC% = 0
ANA%= 16
TIMEOUT = T10s
CALL SendIFC(ISC%)
   IF (ibsta% and EERR) Then Call gpiberr
CALL ibSRE(ISC%,1)
   IF (ibsta% and EERR) Then Call gpiberr
CALL ibTMO(ISC%, TIMEOUT)
   IF (ibsta% and EERR) Then Call gpiberr
CLS

  CALL IOREMOTE(ANA&)
  CODE$ = "TALKLIST;"
  CALL IOOUTPUTS(ANA&, CODE$, LEN(CODE$))
  CODE$ = "IDN?"
  CALL IOOUTPUTS(ANA&, CODE$, LEN(CODE$))
  RCV$ = SPACE$(72)
  ACT% = 0
  MAX% = 72
  CALL IOENTERS(ANA&, RCV$, MAX%, ACT%)
  PRINT "ANA MODEL "; RCV$
  CODE$ = "OPC;PRES;"
  CALL IOOUTPUTS(ANA&, CODE$, LEN(CODE$))
  END

IOREMOTE(ANA%)
   DIM addrlist%(2)
   addrlist%(0) = ANA%
   sddrlist%(1) = NOADDR
   Call EnableRemote(ISC%, addrlist%())
RETURN

IOOUTPUTS(ANA%, CODE$, LEN)
   Send(ISC%, ANA%, CODE$, NLend%)
RETURN

IOENTERS(ANA%, RCV$, MAX%, ACT%)
   Term% = 10
   Receive(ISC%, ANA%, RCV$, Term%)
RETURN
```

Figure 3 Subroutine Method Program Listing