

IEEE 488

APPLICATION BULLETIN

RUNNING AN ICS 8065 FROM THE APPLE OS X OPERATING SYSTEM

INTRODUCTION

With the resurgence in the popularity of Apple's computers, people are using them more often in engineering applications. Since the Apple OS X operating system is a Unix based system, Apple computers are easy to use with servers like ICS's 8065 Ethernet to GPIB Controller that use Remote Procedure Calls (RPC). This application bulletin shows how to quickly install the VXI-11 RPC libraries in an Apple computer and how to develop a simple C program.

VXI-11 INTRODUCTION

ICS's 8065 Ethernet to GPIB Controller (or GPIB Gateway) is a VXI-11 compliant server. The VXI-11 Specification specifies a protocol for communication with (test or measurement) devices over a network (LAN) via TCP/IP. This protocol uses the ONC/RPC (Open Network Computing/Remote Procedure Call) standard which is based on TCP/IP and is described in the 'VXI-11: TCP/IP Instrument Protocol Specification'.

VXI-11 is the overall VXI-11 document and describes the network protocol. There are three sub-specifications. VXI-11.1 is for a VXI chassis and is not applicable to the 8065. The VXI-11.2 Specification defines how the gateway (8065) controls the GPIB bus and performs bus specific functions. The VXI-11.3 Specification describes instrument specific functions including data transfer to and from an instrument.

ICS's 8065 has both VXI-11.2 and VXI-11.3 capability so it can be used just as you would any GPIB Controller with 488.2 capability. Generating IFCs and performing 488.2 protocols such as FindLstn are no problem for the 8065. Competitive units with only VXI-11.3 capability are limited to just device specific functions.

PREPARATION

The first step is the conversion of the RPCL in the VXI-11 Specification to the RPC files specific to your Apple computer. Download a copy of the VXI-11 Specification from ICS's website (www.icselect.com) or from the VXIbus Consortium's website (www.vxibus.org).

Copy the RPCL in the VXI-11 Specification to a `vx11.x` file on your MAC.

Install Apple's Developer Tools on your MAC. You can obtain the Developer Tools from Apple's website.

CREATING THE RPC FILES

Perform the following steps to generate the RPC files. Apple's OS X 10.5 includes an `rpcgen` utility that does the RPC file generation.

1. Open a Terminal Session.
2. Navigate to the folder with the `vx11.x` file.
3. Type `rpcgen vx11.x`

This will generate the following files: `vx11_clnt.c`, `vx11_svc.c`, `vx11_xdr.c` and `vx11.h`. You will not use the `vx11_svc.c` file in this client application.

DEVELOPING THE PROGRAM

1. In Xcode (part of the Apple Developer Tools) create a new project (Command Line utility > Standard Tool).
2. Add the `vx11_clnt.c`, `vx11_xdr.c` and `vx11.h` files to the project.
3. Write your `main.c` file.

RPC programming requires that you first create a link to a device before you can send it commands or read data back from it. When done with the program you destroy the link before exiting the program. The recommendation in the 8065 manual is that you create a link to the 8065 and to each instrument as part of the initialization phase of your program. These links should remain open until you exit the program.

Simple examples like the following one have an open-write-read-close sequence that often confuses users since they are lead to believe that open-write-read-close is the normal command sequence. This is a problem with a short example. Opening and closing links is very time intensive and causes the 8065 to quickly run out of resources.

EXAMPLE PROGRAM

The following example was written by Bela Farago from Institute Laue-Langevin in Grenoble, France. The program is a simple one that does an IDN query of the instrument. Bela wrote it to verify the 8065's operation with the MAC before writing a larger program.

The setup is shown in Figure 1. The instrument used was a Keithley voltmeter which was set to a GPIB address of 30. The program includes calls to `Microseconds()` to time the read, write and `create_link` operations. These calls should be removed from your actual program. The execution times Bela measured with an Intel core-duo mac are:

```
create_link - 265 ms
write-read - 65 ms
close link and destroy client - 20 ms
```

The measured times reinforces our recommendation to create the links to all of your instruments at the start of your program and to leave them open until you end the program. Link creation and destruction are major time sinks.

EXAMPLE MAIN.C

```
#include <stdio.h>
#include <rpc/rpc.h>
#include <string.h>
#include <CoreServices/CoreServices.h>
#include "vxi11.h"

#define WAITLOCK_FLAG 1
#define WRITE_END_CHAR 8
#define READ_END_CHAR 0x80
#define VXI_ENDW (WAITLOCK_FLAG | WRITE_END_CHAR)
```

```
CLIENT *rpcClient;
Create_LinkParms crlp;
Create_LinkResp *crlr;
Device_WriteParms dwrp;
Device_WriteResp *dwrr;
Device_ReadParms drdp;
Device_ReadResp *drdr;
Device_Error *derr;
```

```
UnsignedWide microTickCount1,microTickCount2;
```

```
static char *svName = "192.168.1.254";
```

```
int main(){
char sendMessage[1024];
//char responseMessage[2048];
```

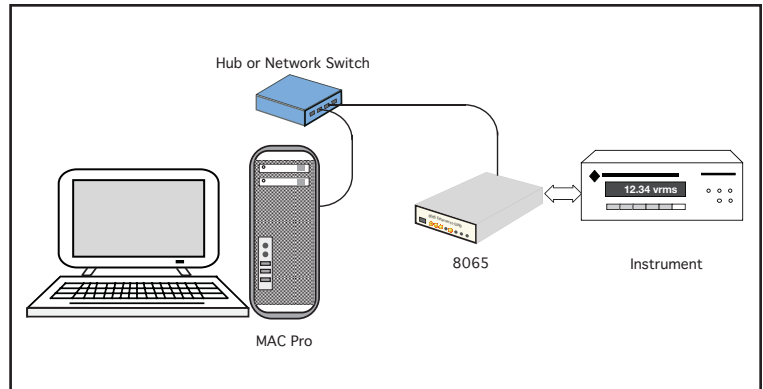


Figure 1 Example Test Setup

```
rpcClient = clnt_create(svName, DEVICE_CORE,
    DEVICE_CORE_VERSION, "tcp");
if (rpcClient == NULL){
    printf("Error creating client\n");
    clnt_pcreateerror(svName);
    return 1;
}
// printf("Hello %s\n", svName);
crlp.clientId = (long) rpcClient;
crlp.lockDevice = 0;
crlp.lock_timeout = 10000;
crlp.device = "gpib0,30";
crlr = create_link_1(&crlp, rpcClient);
if (crlr == NULL){
    clnt_perror(rpcClient, svName);
    return 1;
}
// printf("Link created to %s\n", crlp.device);

Microseconds (&microTickCount1);

dwrp.lid = crlr->lid;
dwrp.io_timeout = 1000;
dwrp.lock_timeout = 10000;
dwrp.flags = VXI_ENDW;
sprintf(sendMessage, "*IDN?\n");
dwrp.data.data_len = strlen(sendMessage);
dwrp.data.data_val = sendMessage;
dwrr = device_write_1(&dwrp, rpcClient);
if (dwrr == NULL){
    clnt_perror(rpcClient, svName);
    printf ("device_write returned dwrr ==
        NULL \n");
    return 1;
}
drdp.lid = crlr->lid;
drdp.io_timeout = 1000;
drdp.lock_timeout = 10000;
drdp.flags = VXI_ENDW;
drdp.termChar = '\n';
```

```

drdp.requestSize = 1024;
drdr = device_read_1(&drdp, rpcClient);
if (drdr == NULL){
    clnt_perror(rpcClient, svName);
    printf("device read returned drdr == NULL\n");
    return 1;
}
// printf("response = %s\n", drdr->data.data_val);

Microseconds ( &microTickCount2);

derr = destroy_link_1(&(crlr->lid), rpcClient);
clnt_destroy(rpcClient);
printf("Elapsed time in microsec %ld\n", microTick-
    Count2.lo-microTickCount1.lo);
return 0;
}

```

SUMMARY

This Application Note has shown how ICS's Model 8065 Ethernet to GPIB Gateway can be controlled from an Apple computer using a program developed in Apple's Xcode. The MAC RPC file generation is easily done using the OS X's built-in rpcgen utility to convert the VXI-11 RPCL into MAC specific files.

The Application Note also includes an Xcode example that does the IEEE-488.2 *IDN query.

CREDITS

Special thanks to Bela Farago at Institut Laue-Langevin, BP:156, 38042 Grenoble Cedex 9, France for the program example and comments about using the 8065 with a MAC.

References:

References:

The following references were used in the course of developing this program:

- * Model 8065 Ethernet <-> GPIB Controller Instruction Manual, ICS Electronics.
- * Frequently Asked Questions (FAQ) For The ICS Model 8065 Ethernet-To-GPIB Controller, Application Note AB80-1, ICS Electronics.
- * VXI-11 RPC Programming Guide for the 8065: An Introduction to RPC Programming, Application Note AB80-3, ICS Electronics.
- * VMEbus Extensions for Instrumentation: TCP/IP Instrument Protocol Specification VXI-11, Revision 1.0, July 17, 1995.
- * Power Programming with RPC, John Bloomer, O'Reilly & Associates, Inc., 1992.