

APPENDIX A EXAMPLE PROGRAM

The following program is a tested example of both HSD and IBL operating modes and is provided as an aid for software development purposes only. In IBL mode, this example is for a PCIHSD board connected to a Encore HSDII board or other compatible interface (i.e., an IBL mode configuration). The PCIHSD initiates all transfers. In HSD mode, the PCIHSD board is connected to an HSD device. This program is provided as part of the distribution package and can be modified by the user as necessary. Applied Data Sciences, Inc. takes no responsibility for any user modification.

```
// testhsd.cpp : Defines the entry point for the console application.
//

#include <windows.h>
#include <winioctl.h>
#include <conio.h>
#include "stdafx.h"

// include PCIHSD definitions
#include <pcihsd.h>
// (pcihsdioctl.h is included by pcihsd.h)

//
// Forward declarations of local functions...
//
static VOID ErrorMessage( LPTSTR lpOrigin, DWORD dwMessageId);
static VOID TestHSDMode(char read_write);
static VOID TestIBLMode(char read_write, char cable_connect);
static VOID dump_iocb( int i, IOCB *iocb);

#define BSIZE (16)
#define NUM_IOCBS (10)

char cDeviceName[] = "\\.\.\PCIHSD0"; // resolves to \.\.\PCIHSD0

void usage( void )
{
    printf("parameter 1 must be 'R' for IBL read transfer\n");
    printf("                or 'W' for IBL write transfer\n");
    printf("                or 'H' for HSD transfer\n");
    printf("                use lower case for continuous execution\n");
    printf("parameter 2 must be 'N' for normal cable connection. (default)\n");
    printf("                or 'R' for reverse cable connection.\n");
}

void main(int argc, char* argv[])
{
    char cCableConnect;
```

```

if( argc > 2 )
{
    cCableConnect = 'N';
    if( *argv[2] == 'R' )
        cCableConnect = 'R';

    if( toupper(*argv[1]) == 'R' )
    {
        printf("Testing Read IBL Mode of PCIHSD Board\n");
        TestIBLMode( *argv[1], cCableConnect );
    }
    else if( toupper(*argv[1]) == 'W' )
    {
        printf("Testing Write IBL Mode of PCIHSD Board\n");
        TestIBLMode( *argv[1], cCableConnect );
    }
    else if( toupper(*argv[1]) == 'H' )
    {
        printf("Testing HSD Mode of PCIHSD Board\n");
        TestHSDMode(*argv[1]);
    }
    else
    {
        printf( " ???  invalid command\n\n" );
        usage();
    }
}
else
{
    usage();
}

ExitProcess( ERROR_SUCCESS );
}

```

```

static VOID TestHSDMode(char read_write)
{
    HANDLE hDevice;
    DWORD dwErrorCode;
    DWORD dwBytesReturned, dwLength, dwResetType;
    ULONG iMode, iCurConfig;
    LONG   iNewTimeout, iOldTimeout;
    DWORD  dwBuffer[BSIZE], dwBuff2[BSIZE];
    PCIHSD_STATUS sStatus;

    int          jk, index;
    int          count = 0;
    IOCB        iocb[NUM_IOCBS], bcoi[NUM_IOCBS];

    for( jk = 0; jk<NUM_IOCBS; jk++)
    {
        bcoi[jk].w1.w = iocb[jk].w1.w = -1;
        bcoi[jk].w2.w = iocb[jk].w2.w = -1;
        bcoi[jk].w3.w = iocb[jk].w3.w = -1;
        bcoi[jk].w4.w = iocb[jk].w4.w = -1;
    }

    // Open the device
    hDevice = CreateFile(
                                cDeviceName,

```

```

        GENERIC_WRITE|GENERIC_READ,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL );
if( hDevice == INVALID_HANDLE_VALUE )
{
    dwErrorCode = GetLastError();
    ErrorMessage( "CreateFile", dwErrorCode );
    ExitProcess( dwErrorCode );
}

if( !DeviceIoControl(hDevice, IOCTL_HSD_GET_STATUS,           //Status command
                    NULL, 0,                                //Input Buffer, size
                    &sStatus, sizeof(PCIHSD_STATUS),        //Output Buffer, size
                    &dwBytesReturned,                      //Bytes returned
                    NULL) )                                //No overlapped IO
{
    dwErrorCode = GetLastError();
    ErrorMessage( "GetStatus", dwErrorCode);
    ExitProcess( dwErrorCode );
}

printf( " BoardStatus: %8.8x\n", sStatus.ulBoardStatus);
printf( "Configuration: %8.8x\n", sStatus.ulState);

iNewTimeout = 50;           // set to timeout in milliseconds

ResetTimeout:

if( !DeviceIoControl(hDevice, IOCTL_HSD_SET_FAST_TIMEOUT, //Fast timeout command
                    &iNewTimeout, sizeof(iNewTimeout),    //Input Buffer, size
                    &iOldTimeout, sizeof(iOldTimeout),    //Output Buffer, size
                    &dwBytesReturned,                    //Bytes returned
                    NULL )                                //No overlapped IO
{
    dwErrorCode = GetLastError();
    ErrorMessage( "DeviceIoControl", dwErrorCode );
    CloseHandle( hDevice );
    ExitProcess( dwErrorCode );
}

printf( "Old Timeout was %ld seconds\n", iOldTimeout);

// set mode of PciHsd board
iMode = PCIM_HSD;           // set HSD mode
iMode |= PCIM_CNRM;        // add in Normal cable connections
iMode |= PCIM_NSXAP;       // add in No byte swap

if( !DeviceIoControl(hDevice, IOCTL_HSD_SET_MODE,         //Mode command
                    &iMode, sizeof(iMode),              //Input Buffer
                    &iCurConfig, sizeof(iCurConfig),   //Output Buffer
                    &dwBytesReturned,                   //Bytes returned
                    NULL )                                //No overlapped IO
{
    dwErrorCode = GetLastError();
    ErrorMessage( "DeviceIoControl", dwErrorCode );
    CloseHandle( hDevice );
    ExitProcess( dwErrorCode );
}

printf( "iMode:      %8x\n", iMode);

```

```

printf( "iCurConfig: %8x\n", iCurConfig);

dwResetType = RESET_BOTH;    // = 0; defaults to RESET_BOTH

if( !DeviceIoControl(hDevice, IOCTL_HSD_RESET,    //Reset command
                    &dwResetType, sizeof(dwResetType), //Input Buffer, size
                    NULL, 0,                    //Output Buffer, size
                    &dwBytesReturned,
                    NULL))
{
    dwErrorCode = GetLastError();
    ErrorMessage( "DeviceIoControl", dwErrorCode );
    CloseHandle( hDevice );
    ExitProcess( dwErrorCode );
}

printf("Device Reset - dwResetType: %d\n",dwResetType);

index = 0;

// create IOCB list
iocb[index].w1.f.hsd_cmd = IOc_CMD | IOc_CC; //device command with command chain
iocb[index].w1.f.udd_cmd = 0x01;           //set user byte
iocb[index].w1.f.count = 0;
iocb[index].w2.w = 0x87654321;
iocb[index].w3.w = 0;
iocb[index].w4.w = 0;
index++;

iocb[index].w1.f.hsd_cmd = IOc_RD | IOc_CC; // read data with command chain
iocb[index].w1.f.udd_cmd = 0x01;
iocb[index].w1.f.count = BSIZE;
iocb[index].w2.a = &dwBuffer[0];
iocb[index].w3.w = 0;
iocb[index].w4.w = 0;
index++;

iocb[index].w1.f.hsd_cmd = IOc_RD | IOc_CC; // read data with command chain
iocb[index].w1.f.udd_cmd = 0x01;
iocb[index].w1.f.count = BSIZE;
iocb[index].w2.a = &dwBuff2[0];
iocb[index].w3.w = 0;
iocb[index].w4.w = 0;
index++;

iocb[index].w1.f.hsd_cmd = IOc_DSR;        // device status request
iocb[index].w1.f.udd_cmd = 0x01;
iocb[index].w1.f.count = 0;
iocb[index].w2.w = 0;
iocb[index].w3.w = 0;
iocb[index].w4.w = 0x5a5a5a5a;
index++;

// send IOCB List
dwLength = sizeof( IOCB ) * index;

do
{
    // send the IOCB list for execution
    if( !DeviceIoControl(hDevice, IOCTL_HSD_START, //IO command
                        &iocb[0], dwLength,      //Input Buffer, size
                        &bcoi[0], dwLength,      //Output Buffer, size
                        &dwBytesReturned,
                        NULL ))
    {

```

```

        dwErrorCode = GetLastError();
        ErrorMessage( "DeviceIoControl", dwErrorCode );

        // check on fast timeout value
        if( iNewTimeout )
        {
            // if still on fast timeout, reset it, and try again
            iNewTimeout = 0;
            goto ResetTimeout;
        }

        // close the device and exit
        CloseHandle( hDevice );
        ExitProcess( dwErrorCode );
    }

    // if lower case, check for exit command
    if( read_write == 'h' )
    {
        printf( "%d\t\t\tcycles\r", ++count);
        if( _kbhit() )
        {
            read_write = 'H';
            _getch();
            printf("\n");
        }
    }
    // loop as long as read_write code is lower case H
}
while( read_write == 'h' );

// dump buffers
for( jk=0; jk<BSIZE; jk++)
{
    printf("%3d: %8.8x %8.8x\n", jk, dwBuffer[jk], dwBuff2[jk]);
}

printf( "Success !\n" );

printf( "device status = %4.4x\n", bcoi[index-1].w4.f.sts);
printf( "device count = %4.4x\n", bcoi[index-1].w4.f.cnt);
printf( "dwLength: %d dwBytesReturned: %d index: %d\n",
        dwLength, dwBytesReturned, index);

// dump original IOCB list
printf("\nIOCB List original\n");
for( jk=0; jk<index; jk++)
{
    dump_iocb( jk, &iocb[jk]);
}

// dump returned IOCB list
printf("\nIOCB List returned\n");
for( jk=0; jk<index; jk++)
{
    dump_iocb( jk, &bcoi[jk]);
}

// Close the device
CloseHandle( hDevice );
}

static VOID dump_iocb( int i, IOCB *iocb)
{
    printf("%3d: %8.8x", i, iocb->w1.w);
    printf(" %8.8x", iocb->w2.w);
    printf(" %8.8x", iocb->w3.w);
}

```

```

        printf("    %8.8x\n", iocb->w4.w);
    }

static VOID TestIBLMode(char read_write, char cable_connect)
{
    HANDLE hDevice;
    DWORD dwErrorCode;
    DWORD dwBytesReturned, dwLength;
    ULONG iMode, iCurConfig;
    DWORD dwBuffer[BFSIZE];

    int          jk, index;
    int          count=0;
    IOCB        iocb[NUM_IOCBS],bcoi[NUM_IOCBS];

    // fill buffer in case of write
    for( jk=0; jk<BFSIZE; jk++)
        dwBuffer[jk] = 0x12345600 +jk;

    // Open the device
    hDevice = CreateFile(
        cDeviceName,
        GENERIC_WRITE|GENERIC_READ,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL );
    if( hDevice == INVALID_HANDLE_VALUE )
    {
        dwErrorCode = GetLastError();
        ErrorMessage( "CreateFile", dwErrorCode );
        ExitProcess( dwErrorCode );
    }

    // set mode of PciHsd board
    iMode = PCIM_IBL; // set IBL mode

    if( cable_connect == 'R' )
        iMode |= PCIM_CREV; // add in Reverse cable connections
    else
        iMode |= PCIM_CNRM; // add in Normal cable connections

    iMode |= PCIM_NSWAP; // add in No byte swap

    iMode |= PCIM_LREQ; // set for link request always for HSD Analyzer/Tester
    // iMode |= PCIM_LIBLG; // set for link based on read/ write code

    if( !DeviceIoControl(hDevice, IOCTL_HSD_SET_MODE,
        &iMode, sizeof(iMode),
        &iCurConfig, sizeof(iCurConfig),
        &dwBytesReturned,
        NULL ))
    {
        dwErrorCode = GetLastError();
        ErrorMessage( "DeviceIoControl", dwErrorCode );
        CloseHandle( hDevice );
        ExitProcess( dwErrorCode );
    }

    printf( "iMode:      %8x\n", iMode);
    printf( "iCurConfig: %8x\n", iCurConfig);

    // set up an IBL IOCB for either Read or Write
    index = 0;

```

```

iocb[index].w1.f.hsd_cmd = (toupper(read_write) == 'R') ? IOC_RD: IOC_WT;
iocb[index].w1.f.udd_cmd = 0x01;
iocb[index].w1.f.count = BSIZE;
iocb[index].w2.a = &dwBuffer[0];
iocb[index].w3.w = 0;
iocb[index].w4.w = 0;
index++;

// send IOCB List
dwLength = sizeof( IOCB ) * index;

do
{
    if( !DeviceIoControl(hDevice, IOCTL_HSD_START,
        &iocb[0], dwLength,
        &bcoi[0], dwLength,
        &dwBytesReturned,
        NULL ) )
    {
        dwErrorCode = GetLastError();
        ErrorMessage( "DeviceIoControl", dwErrorCode );
        CloseHandle( hDevice );
        ExitProcess( dwErrorCode );
    }
    if( (read_write == 'r') || (read_write == 'w') )
    {
        printf(" %d\t\tcycles\r", ++count);
        if( _kbhit() )
        {
            _getch();
            read_write = toupper( read_write );
            printf("\n");
        }
    }
}
while( (read_write == 'r') || (read_write == 'w') );

for( jk=0; jk<BSIZE; jk++)
{
    printf("%3d: %8.8x\n", jk, dwBuffer[jk]);
}

printf( "Success !\n" );

printf("\nIOCB List original\n");
for( jk=0; jk<index; jk++)
{
    printf("%3d: %8.8x", jk, iocb[jk].w1.w);
    printf(" %8.8x", iocb[jk].w2.w);
    printf(" %8.8x", iocb[jk].w3.w);
    printf(" %8.8x\n", iocb[jk].w4.w);
}

printf("\nIOCB List returned\n");
for( jk=0; jk<index; jk++)
{
    printf("%3d: %8.8x", jk, bcoi[jk].w1.w);
    printf(" %8.8x", bcoi[jk].w2.w);
    printf(" %8.8x", bcoi[jk].w3.w);
    printf(" %8.8x\n", bcoi[jk].w4.w);
}

// Close the device
CloseHandle( hDevice );
}

```

```

//
// This helper routine converts a system-service error
// code into a text message and prints it on StdOutput
//
static VOID
ErrorMessage(
    LPTSTR lpOrigin,        // Indicates error location
    DWORD dwMessageId      // ERROR_XXX code value
)
{
    LPTSTR msgBuffer;      // string returned from system
    DWORD cBytes;          // length of returned string

    cBytes = FormatMessage(
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_ALLOCATE_BUFFER,
        NULL,
        dwMessageId,
        MAKELANGID(LANG_ENGLISH, SUBLANG_ENGLISH_US),
        (TCHAR *)&msgBuffer,
        500,
        NULL );

    if( msgBuffer )
    {
        msgBuffer[ cBytes ] = TEXT('\0');
        printf( "Error: %s -- %s\n", lpOrigin, msgBuffer );
        LocalFree( msgBuffer );
    }
    else
    {
        printf( "FormatMessage error: %d\n", GetLastError());
    }
}

```