ICS ELECTRONICS
division of Systems West Inc.

IEEE 488 APPLICATION BULLETIN

ASYNCHRONOUS EVENT NOTIFICATION WITH ICS's USB to GPIB CONTROLLERS

INTRODUCTION

In many application, it is often helpful if the user's program can be notified when an even has occurred. WIN32 GPIB applications can be programmed to receive notification of asynchronous events by using the ibnotify function. The purpose of this application note is to demonstrate how to use the `ibnotify` function in a WIN32 application.

IBNOTIFY

`ibnotify` is a function that can be invoked when any of its enabled mask bits occurs. For device-level calls, valid mask bits are I/O Complete (CMPL), Timeout (TIMO), END or EOS detected (END) and Service Request detected (RQS). Refer to ICS's 488.2 Manual for a complete description of the `ibnotify` function and the board-level mask bits.

`ibnotify` can be called to inform the application when one of these events occurs. A typical application might want to know when a particular device is requesting service. When the device requests service, `ibnotify` invokes a separate (Callback) routine to service the call.

Note: These Callback routines operate in a separate program thread and have access to the global variables. If other GPIB commands may be executed while `ibnotify` is active, then you should use the separate thread functions. In addition use program semaphores to protect other shared program variables. See the `ibnotify` description in the 488.2 Manual.

EXAMPLE

The attached program illustrates using the `ibnotify` command to notify the application when a device receives a serial message. The example setup is shown in Figure 1. The GPIB Controller is attached to an ICS Model 4896 GPIB to Quad Serial Interface. The 4896 has four serial channels that operate with RS-232 or RS-422/RS-485 serial signals. `ibnotify` is used to set a data flag when Channel one of the 4896 receives serial messages.

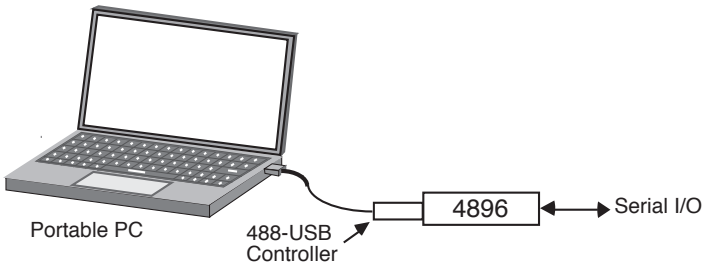

Figure 1 Example Setup

The 4896's four serial channels are addressed with secondary addresses 1, 2, 3 and 4. These serial channels transparently pass data between the GPIB bus and the serial interfaces. The control functions for the data channels are addresses at secondary addresses 11, 12, 13 and 14. These control channels are 488.2 compliant interfaces and use SCPI commands to set the serial parameters.

The 4896's Status Reporting Structure is shown in Figure 2. The 4896 has a complete 488.2 Status Reporting Structure that is augmented with the Questionable Register array and Operation Arrays. The Questionable Condition register reports the current condition of the receive buffers and messages. The Questionable Event register bits are edge sensitive and set when a Condition bit becomes true. If the corresponding Questionable Enable bit is true, then a summary bit is generated which cascades down to bit 3 in the Status Byte register. If the corresponding Status Register Enable bit is set, then the 4896 will generate a Service Request when it receives a message. The example program uses this mechanism to generate a Service

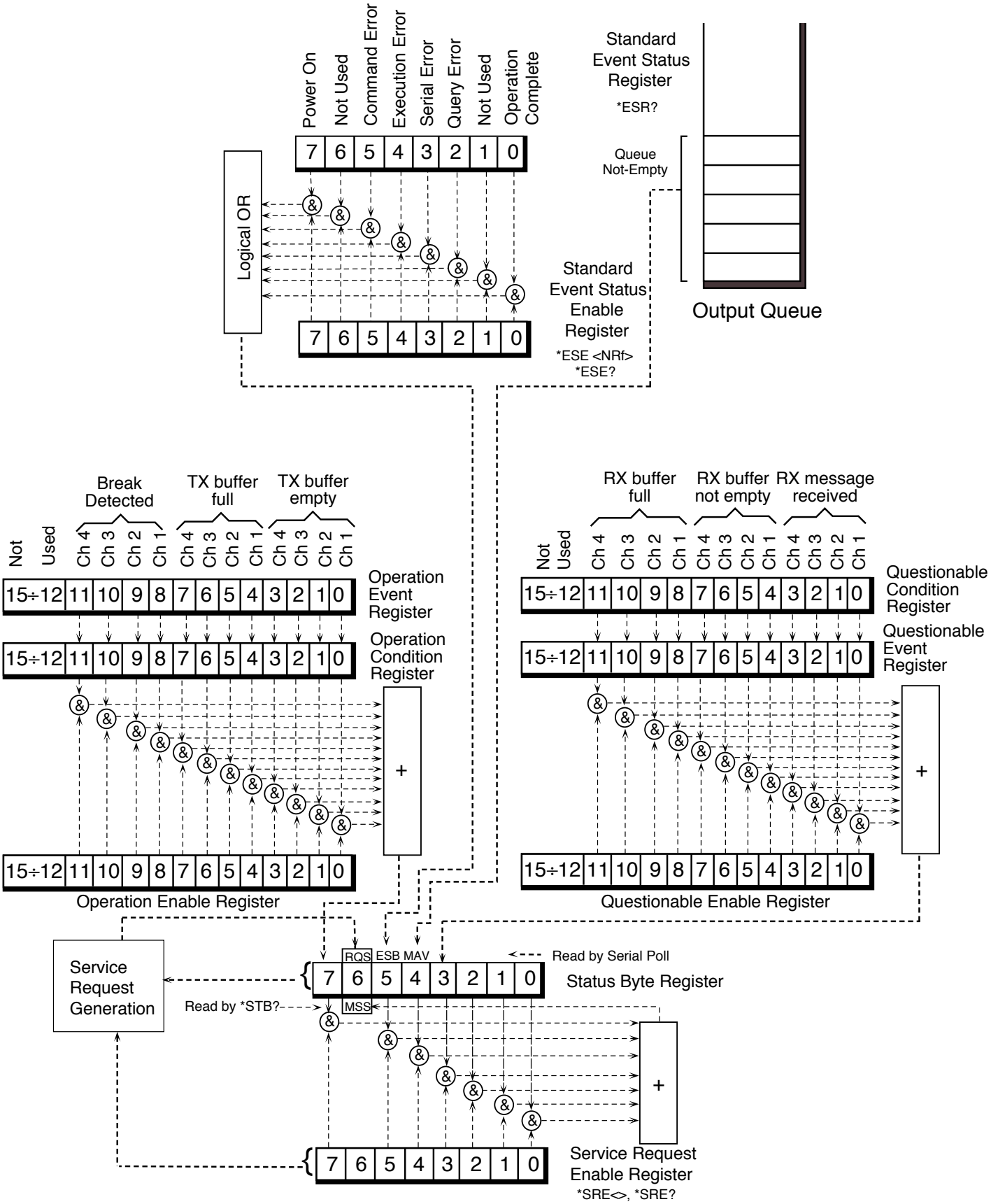

Figure 2 4896's Status reporting Structure

Request each time the 4896 receives a serial massage.

PROGRAM

The listing for the example program is shown in Figure 3. The complete program and supporting files can be downloaded from ICS's website.

The first section defines the commands used to initialize the 4896. Note that the Questionable Register array is enabled to report received messages for channel one. The message terminator is set to carriage return (decimal 13).

The next section uses `ibfind` to obtain the board handle. `ibdev` is then used to get the handles for the 4896's Control and Data channels. The 4896's IDN message is queried and printed on the screen. A Device Clear and a serial poll clear out any left over data in the serial channels and eliminate any 4896 SRQs.

The writes in the following section output the setup commands to the 4896 and clears the Event Status Register. The last line of this section is a call to the `ibnotify` function so that it will return the data flag when a SRQ is generated by the 4896.

The program then cycles through the two if statements until terminated by the user from the keyboard.

```
//
//  TEST PROGRAM FOR notify FUNCTION.
//                  USES ICS 4896 MINI-BOX
//                  M. FRY   9/12/02
//
#include “stdafx.h”
#include <stdlib.h>
#define _X86_
#include <Windef.h>
#include <Winbase.h>
#include <string.h>
#include <conio.h>
#include “ICSDecl.h”
//
int __stdcall MyCallback(int,int,int,long,void *);
//
int main(int argc, char* argv[]) {
        int intfc,handle_CTL,handle_DATA;
        int dataFlag,QEvent;
        char setup1[] = “SYST:COMM:SER:BAUD 9600”;
        char setup2[] = “*SRE 40”;
        char setup3[] = “STAT:QUES:ENAB 1”;
        char setup4[] = “SYST:COMM:SER:EOM 13”;
```

Figure 3 Example ibnotify Program

When the 4896 receives a serial message terminated with a carriage return, the MyCallback routine puts the Serial Poll Response in the return variable. The next time the application tests the dataflag, it executes the statements in the dataflag if routine to and read and print the serial message. The statements also query the 4896's Questionable Event Register to clear it and re serial poll the 4896 to clear its Status Register to prepare it for the next message.

SUMMARY

This application note has described an `ibnotify` example application that reads and prints serial message that are received asynchronously. This example uses an ICS Model 4896 GPIB to Quad Serial Interface to receive the serial massage. A zip file withe the complete `ibnotify` can be downloaded from the Application Notes page on ICS Electronics' website at http://www.icselect.com/.

```
        char setup5[] = “SYST:COMM:SER:ADD:ENAB 0”;
        char setup6[] = “SYST:COMM:SER:EOI ON”;
        char buf[100];
        char QBuf[10];
        char pollByte;
//
        dataFlag = 0;
        printf(“GPIB notify testing\n”);
        intfc = ibfind(“GPIB0”);
        handle_CTL = ibdev(0,4,107,T3s,1,0);
        handle_DATA = ibdev(0,4,97,T3s,1,0);
        ibwrt(handle_CTL,”*IDN?”,5);
        ibrd(handle_CTL,buf,100);
        buf[ibcnt-1] = 0;
        printf(“\nTEST BOX ID: %s”,buf);
        ibwrt(handle_CTL,”*RST”,4);
        Sleep(500);

//      added to clear serial channel, eliminates 4896 hang if data in buffer
        ibclr(handle_DATA);
        Sleep(500);
//      eliminates any existing SRQ’s
        ibrsp(handle_DATA, &pollByte);

        ibwrt(handle_CTL,setup1,strlen(setup1));
        printf(“\nSTEP-1”);
        Sleep(500);
        ibwrt(handle_CTL,setup2,strlen(setup2));
        printf(“\nSTEP-2”);
        Sleep(500);
        ibwrt(handle_CTL,setup3,strlen(setup3));
        printf(“\nSTEP-3”);
        Sleep(500);
        ibwrt(handle_CTL,setup4,strlen(setup4));
        printf(“\nSTEP-4”);
        Sleep(500);
        ibwrt(handle_CTL,setup5,strlen(setup5));
        printf(“\nSTEP-5”);
        Sleep(500);
        ibwrt(handle_CTL,setup6,strlen(setup6));
        printf(“\nSTEP-6”);
        Sleep(500);
        ibwrt(handle_CTL,”*CLS”,4);
        Sleep(500);
        ibrsp(handle_CTL,&pollByte);
        ibnotify(handle_CTL,RQS,MyCallback,(void *)&dataFlag);
```

Figure 3 Example ibnotify Program continued

```
        printf("\r\n\nEnter Serial Message to Test ibnotify: ");
TOP:
        if( kbhit() ) {
                if( getchar() == 0x0D ) goto TOP;
                if( getchar() == 0x0A ) goto TOP;
                if( getchar() == '?' ) {
                        ibrd(handle_DATA,buf,100);
                        if( ibsta & ERR ) {
                                printf("\nREAD ERROR: %d",iberr);
                                }
                        else {
                                if( ibcnt > 0 ) {
                                        buf[ibcnt-1] = 0;
                                        printf("\nINPUT DATA: %s",buf);
                                        }
                                }
                        goto TOP;
                        }
                // Enter two spaces and enter, to exit program
                goto STOP;
                }
        if( dataFlag ) {
                ibwrt(handle_CTL,"STAT:QUES:EVEN?",15);
                ibrd(handle_CTL,QBuf,10);
                QEvent = atoi(QBuf);
                printf("\nQuestionable Events(HEX):%04X",QEvent);
                ibrd(handle_DATA,buf,100);
                buf[ibcnt-1] = 0;
                printf("\n%s",buf);
                dataFlag = 0;
                printf("\r\n\nEnter Serial Message to Test ibnotify: ");
//
//              added, clears up hang up on 4896 status byte register
                ibrsp(handle_DATA, &pollByte);          /* Resets the 4896 status reg */
                }
        goto TOP;
STOP:
        return 0;
}
//
//  ROUTINE TO HANDLE ibnotify CALLBACK.
//
int __stdcall MyCallback(int Ud, int Ibsta,
                                                int Iberr, long Ibcntl, void *RefData) {
        char pollByte;
//
        *(int *)RefData = 1;
        ibrsp(Ud,&pollByte);   /* ??? */
        return(RQS);
        }
```

Figure 3 Example ibnotify Program continued