8019? archive

The University of Michigan

IVHS

Intelligent Vehicle-Highway Systems

Interfacing the Nintendo Power Glove to a Macintosh Computer

Marie Williams
Paul Green

September 1990

IVHS Technical Report-90-14

UNIVERSITY OF MICHIGAN
TRANSPORTATION RESEARCH INSTITUTE • (313) 936-1066
2901 Baxter Road, Ann Arbor, MI 48109-2150, and

COLLEGE OF ENGINEERING • (313) 764-4333
4112 EECS, Ann Arbor, MI 48109-2122

Technical Report Documentation Page

1. Report No. UMTRI-90-36	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle INTERFACING THE NINTENDO POWER GLOVE TO A MACINTOSH COMPUTER		5. Report Date September, 1990 6. Performing Organization Code
7. Author(s) Marie Williams and Paul Green		8. Performing Organization Report No. UMTRI-90-36
9. Performing Organization Name and Address The University of Michigan Transportation Research Institute Ann Arbor, Michigan 48109-2150 U.S.A.		10. Work Unit No. (TRAIS) 11. Contract or Grant No.
12. Sponsoring Agency Name and Address IVHS Program Office University of Michigan Transportation Research Inst. 2901 Baxter Road Ann Arbor, Michigan 48109-2150 U.S.A.		13. Type of Report and Period Covered Interim Report 14. Sponsoring Agency Code IVHS TR-90-14
15. Supplementary Notes Funded by U of M IVHS basic research program Sponsors: GM, Ford, Motorola, MDOT, NHTSA		

16. Abstract

This article describes the software and hardware necessary to interface a Nintendo Power Glove to a Macintosh computer. Located on top of the hand are two speakers whose ouput is received by an array of three microphones. Right hand location (to the nearest 1/4 inch) and orientation are determined by three dimensional triangulation of the sound signals. In addition, independent information on the state of all fingers on that hand except the little finger (flexed or extended) is provided.

The glove is normally used as the input device to a video game. In this case the device serves as a sensor of driver hand position in human factors studies on the use of automobile controls.

The hardware interface described controls the Power Glove's signal lines and converts the data from the Glove into RS-422 for the Macintosh serial port. The interface samples the Glove's position under control of a C program (Appendix B).

In-vehicle tests have shown that some filtering of the data is required. When the glove is not pointed towards the sensors the ultrasonic pulse from the glove is reflected off of the windshield or instrument panel and the location of the glove is incorrectly identified. This is easily corrected.

17. Key Words Human factors, ergonomics, human engineering, engineering psychology, human movement	18. Distribution Statement		
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 20	22. Price

CONTENTS

METHODS FOR EXAMINING THE USE OF CONTROLS

Knowledge of how and when controls are used is essential in designing controls for vehicles of all types. (See Turner and Green, 1987 for a review of automobile control literature.) This information can provide information on frequency and sequence of use on which new designs can be based and critiqued. In addition, such information may be desired for more basic studies of human movement and efforts to model human performance.

Data on control use can be collected by various means--experimenter observation, driver estimation, instrumentation of the vehicle, or instrumentation of the driver. The simplest means of recording control use is to have a trained experimenter either watch drivers directly or look at videotapes of control use. While direct observation is easy to do and provides accurate frequency data, the time durations associated with control use are not very accurate. Added accuracy can be obtained by videotaping drivers and then playing back the tapes, but this is time-consuming. Furthermore, the data quality depends on having good lines of sight to the controls, which is not always possible.

Asking drivers how often they use various controls does not lead to reliable values. Driver estimates are particularly poor for controls that are used infrequently. How often various controls are used often depends on context (for example, the defroster is generally used only in the winter) and it is difficult for people to accurately estimate the usage of these controls. Also, driver estimates do not provide information on the timing associated with control use.

Instrumenting the vehicle might seem like an obvious approach, but connecting each switch of interest to an interface box and writing a software logging routine is both expensive and time-consuming. For example, to record the use of a contemporary cassette-radio unit would require at least three 8 bit I/O ports, and that is but one of many secondary controls in a car. While the recording of control use data will be easier when a bus-type architecture replaces the point-to-point wiring in automobiles, this will not occur for some time.

Instead of monitoring control use directly, an alternative is to monitor driver activity. Sensors of various types--magnetic, radio frequency (RF), visual, or acoustic--can be placed on the driver's hands. But magnetic sensors tend not to work very well inside motor vehicles, and the noisy RF environment of an automobile makes sensors of this type imprecise. Visual systems, which generally involve automated videotape analysis either require specialized software, specialized and expensive hardware, or both (e.g., the Selspot system).

The best known commercial product for measuring hand position is the VPL DataGlove (Conn, Lanier, Minsky, Fisher, and Druin, 1989; Foley, 1990; Weimer and Gamapathy, 1989; Zimmerman, Lanier, Blanchard, Bryson, and Harvil, 1987). The DataGlove uses a polhemus magnetic sensor to determine hand position and a goniometer for each finger joint to determine its angle. This device is extremely accurate. One convincing demonstration involves a person pretending to play a guitar (the "air guitar"). A computer linked to the DataGlove determines the chord and plays it. But a DataGlove set (two sizes cover the population range) and supporting software typically costs about $25,000, an amount well beyond the budget of most researchers.

Commercial acoustically-based systems for recording limb positions, such as the Science Accessories Corporation GP8-3D Sonic Digitizer tend to be somewhat less expensive (just over $15,000), but nonetheless require specialized software for each application.

NINTENDO POWER GLOVE

An interesting and inexpensive alternative is the Power Glove, a device used with Nintendo video games. The Power Glove costs only about $80, and comes in two sizes, small/medium and large. The best known game, Punchout, allows a user to box with a character shown on a video display.

This paper describes the hardware and software needed to interface the Power Glove to a Macintosh computer. This interface allows the Macintosh to sample the Power Glove's position at any interval specified by software.

The Power Glove has two modes, standard joystick mode and absolute mode (also referred to as virtual mode). Joystick mode is used with the Nintendo video games. Absolute mode returns absolute position, but is accessible only by means of a proprietary code sent to the glove through one of its signal lines.

A diagram of the Nintendo Power Glove setup is shown in Figure 1. The glove can either provide information on its absolute position (location in space) or relative motion (like a joystick). In addition to hand position, the Glove also independently senses the position of each finger except the little finger.

To determine its location and orientation, the Power Glove emits an ultrasonic pulse from one of two transmitters located on the Glove unit. The Glove measures the time delays between its transmission of a pulse and the reception by each of three receivers in the sensor array. Using the speed of sound in air, the Glove position is computed via three-dimensional triangulation. The difference in the locations of the transmitters is used to compute the rotation (roll) of the hand.

The reported lateral resolution (Eglowstein, 1990) of the device is highest, one quarter of an inch, when the Glove is equidistant from each of the ultrasonic receivers, and decreases as the Glove moves from this center. The Glove does not function reliably outside of "the sensing zone," the zone about five feet directly in front of the sensor assembly. This is due to the directionality of the transmitters and receivers, so it will track outside of this zone if the Glove's transmitters are aimed at the sensor assembly.

The glove senses finger position by the charge collected on an internal capacitor (one for each finger). A conductive strip embedded in the plastic shell above each finger increases in resistance as it is bent, reducing the current to the capacitor which is discharged periodically to determine the position of the finger.

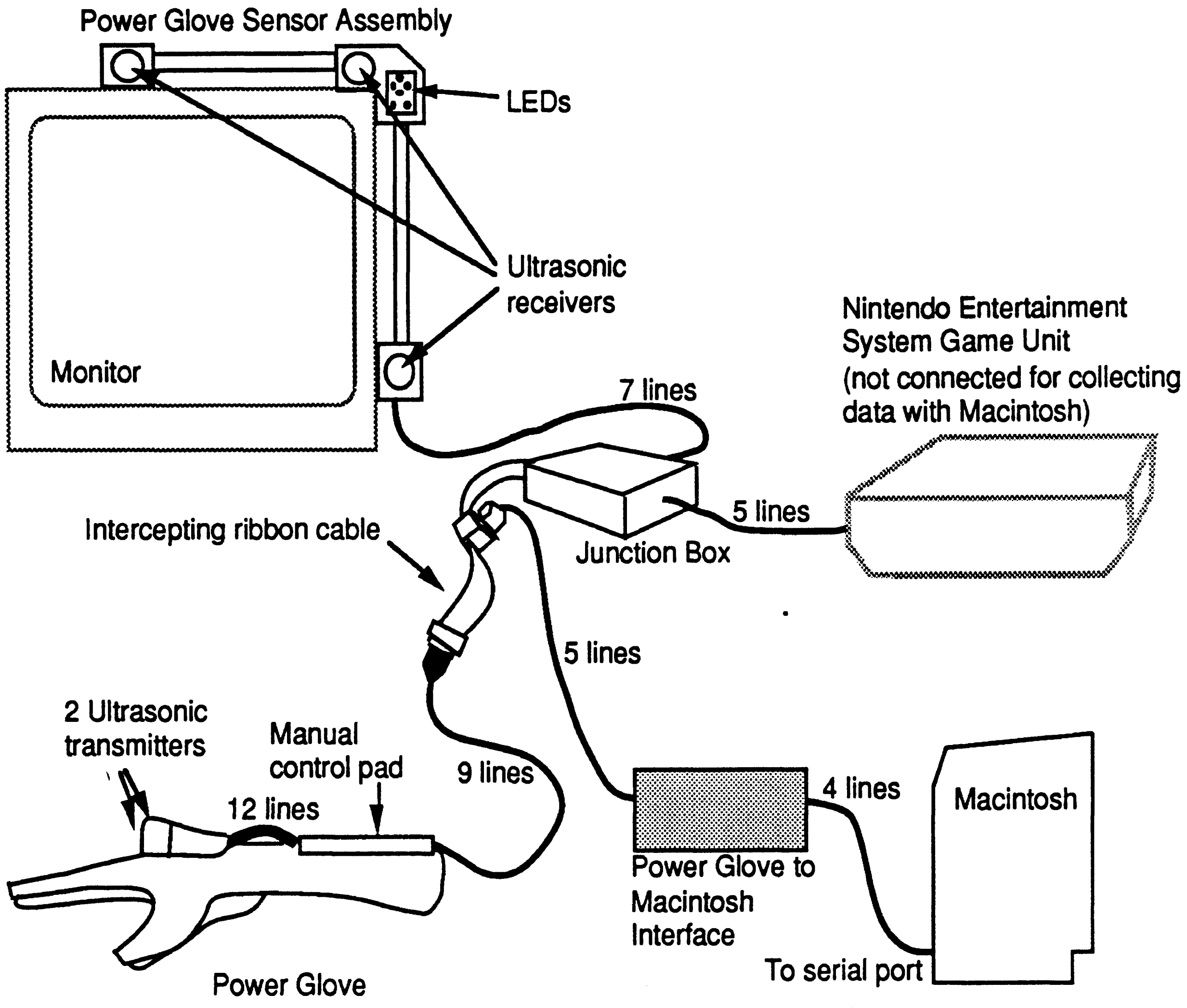

Power Glove Sensor Assembly
LEDs
Ultrasonic receivers
Monitor
Nintendo Entertainment System Game Unit (not connected for collecting data with Macintosh)
7 lines
5 lines
Intercepting ribbon cable
Junction Box
5 lines
2 Ultrasonic transmitters
Manual control pad
9 lines
12 lines
4 lines
Macintosh
Power Glove to Macintosh Interface
To serial port
Power Glove

Figure 1. Nintendo Power Glove System Setup.

Description of the Glove Signals

The Glove and Nintendo Game Unit communicate through five lines, three signal lines plus power and ground. (See Table 1.) The Game Unit samples the status of the Glove (or joystick) by pulling the Latch line high. (See Figure 1.) Triggered by the rising edge of this line, the Glove responds by putting the first data bit on the Data Out line. The Nintendo game then sends the Glove eight pulses which the Glove uses to shift the remaining seven data bits out onto the Data Out line. The Glove is slave to this signal and its rate can be safely decreased from the approximately 85 Kbaud the Nintendo Game Unit uses down to a more workable 9600 baud.

The data bits arrive as shown in Figure 2. These data bits relate to the original Nintendo Entertainment System Controller, where "A" and "B" are buttons used for various video game functions such as fire buttons. On the Power Glove these bits usually correspond to bending of the thumb, "A," and index finger, "B." The "select" button is used for altering game options and the "start" button activates the game and the Glove itself. The remaining directional bits correspond to deflection of the glove from center. A signal is active when the bit is low, so a centered glove with all fingers straight would send a data byte of all high bits (all 1s).

Table 1
Power Glove/ Nintendo Game Connections

Pin	Signal
1	Ground
2	Data Clock
3	Latch
4	Data Out
5	Not used
6	Not used
7	Power +5 volts

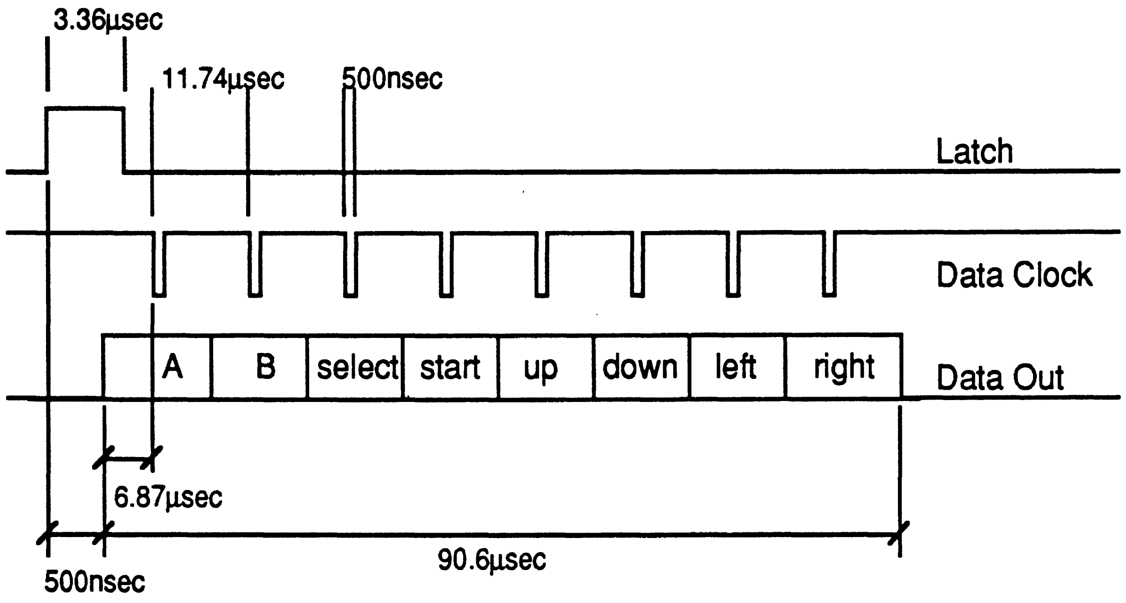


Figure 2. Power Glove/Nintendo Entertainment System Signals.

The Power Glove has 14 built-in programs which are described in detail in the Power Glove Instructions (Mattel, Inc., 1989). Each program has a different set of hand motions to produce the standard joystick signals. In the current configuration, the Glove can only return directional information relative to its center, and different finger position and hand roll information depending on the program being used.

Hardware Required

The hardware necessary to connect the Power Glove to a Mac Plus, Mac SE, or any Mac II series computer, is considerably more complicated then that necessary for connection to an IBM PC or clone. On a PC, all that is needed is a change of connectors and a device driver such as the one listed in Eglowstein (1990). Device drivers are extremely difficult to write for the Macintosh, so the authors have chosen to construct hardware to drive the Glove signals and to pass the resulting data to the Macintosh serial port in RS-422 format.

The interface circuit (in Appendix A), is located in the Power Glove to Macintosh Interface box shown in Figure 1. The circuit contains a PAL (Programmable Array Logic chip), a clock chip and crystal, a 4-bit counter, some basic gates, latches, and an RS-422 driver and receiver. The circuit is controlled by a simple state machine programmed onto the PAL chip which takes its input signals from the 4-bit counter. The input signals and corresponding output signals for each state are listed in Table 2. A data byte is sampled from the Glove whenever the counter input to the PAL is enabled at the count of zero. The counter is enabled by de-asserting the HSKo (Handshake Out) signal from the Mac, which is easily done in software. The counter is cleared and disabled after reaching the count of eleven, and at the next HSKo pulse, will begin counting again at zero. As the interface receives each data bit from the Glove, it latches it and passes it through an RS-422 compatible driver to the Mac at the baud rate specified by the 4702 chip in the circuit.

Table 2
Programmable Logic Truth Table

counter	signal					
state	counten	glove clken!	latch	startbit!	stopbit+!	macserial
0000	0	0	0	0	1	high
0001	1	0	1	0	1	high
0010	1	0	0	1	0	startbit
0011	1	1	0	0	0	bit 0
0100	1	1	0	0	0	bit 1
0101	1	1	0	0	0	bit 2
0110	1	1	0	0	0	bit 3
0111	1	1	0	0	0	bit 4
1000	1	1	0	0	0	bit 5
1001	1	1	0	0	0	bit 6
1010	1	1	0	0	0	bit 7
1011	1	0	0	0	1	stopbit
1100	0	0	0	0	0	
1101	0	0	0	0	0	
1110	0	0	0	0	0	
1111	0	0	0	0	0	

Note--1 means asserted
0 means de-asserted
! indicates negated signal

The easiest way to connect the interface to the Power Glove is by constructing a short cable to intercept the Power Glove signals between the Junction Box and the Glove itself, leaving the cable that would normally plug into the Nintendo Game Unit disconnected. This straight-through cable consists of a nine contact ribbon cable with a male DB-9 on one end, a female DB-9 on the other end, and another DB-9 connector tap in the middle. The gender of the middle connector depends on the gender used on the cable connecting the interface board.

The glove can be made less cumbersome by removing the arm portion of the Glove. By removing the four screws under the manual control pad of the Glove and adding another few feet of cable to extend the 12 wire cable connecting the transmitter portion of the Glove to the manual control pad, the rubber arm portion of the Glove can be cut away making the system more manageable for the user.

Software

The software listed in Appendix B is a generic data collection program written in Think C (Symantec Corporation, 1989). The program calls the minimum Macintosh system toolbox initialization routines and de-asserts HSKo at the desired data collection interval, sending the data (in binary) to the screen and a file called "output." The screen data are a column of 8 bit binary numbers with the bit definitions (zero is active) from left to right as follows: right, left, down, up, start, select, B, A. The program collects data as soon as it is run and terminates when the mouse button is clicked. The program's purpose is to test and demonstrate the hardware, and to provide a base on which a more rigorous and easier to use data collection program can be written.

A demo program for HyperCard is listed in Appendix C. It uses external commands (XCMDs) available through the Apple Developers Association, and four hand icons (all fingers straight, thumb bent, fingers bent, fingers and thumb bent) created using Icon Factory, Hyperpress Publishing Corporation (1988). This program samples the serial port, converts the ASCII into binary and then tests the "A" and "B" bits of the character and presents the hand icon representing the proper finger positions. The program then tests the directional bits to determine which of nine regions (center, up, down, right, left, up-right, up-left, down-right, down-left) the Glove is in and moves the hand icon to that region.

In-Vehicle Tests

The Glove and Macintosh set-up was tested in a 1981 Chevrolet Caprice station wagon. The Glove was not disturbed by the background noise of the automobile, and provided ample coverage of the right half of the steering wheel and center console.

When the Glove leaves "the sensing zone" or aims in a direction other than that of the sensor assembly, the enclosed space makes erroneous data more likely since ultrasonic reflections from the many close surfaces (panel surface, windshield) are strong enough to be interpreted as line-of-sight transmissions by the Glove. In the future with virtual mode functioning, these data will be easy to filter in software since the distance travelled by a reflected transmission is enough farther than a genuine line-of-sight transmission as to make the data unreasonable (the driver seems to have jumped into the back seat).

The main concern when setting up in a car is assuring direct line-of-sight for the Glove to all three receivers, and arranging the sensor assembly such that it is mounted farther back than the farthest control of interest. So, for example, in the test vehicle with the tubing intact, it was quite easy for a target of interest (e.g., the radio) to be behind the plane of the sensing array. It may be necessary to disconnect or even remove the spacing tubing from the sensor assembly to allow for the desired installation. The 3-1/2 inches of extra wire between the receivers allows for some degree of placement freedom. It is important that the final mounted center to center spacing of the receivers be 16 inches and that they retain the right-angle, flat-plane positions of the original assembly.

Future Developments

The obvious next step is adding the capability for obtaining absolute position information from the Glove. VPL Research of Redwood City, California plans to supply an interface box which will contain the proprietary codes necessary to force the Glove to return absolute information. This box will be available only to academic institutions with the next few months. There are plans to offer a low cost DataGlove based on the Power Glove technology by the summer of 1991.

Transfinite Systems Company, Inc. of Cambridge, MA currently offers the "Gold Brick," an ADB (Apple Desktop Bus) compatible Power Glove interface for the Macintosh. This interface allows the Glove to replace the Macintosh mouse, and includes in it's software a demo program in which the user can do manipulations of a cube in three dimensions. This interface "fakes" the z-dimension by using the "A" (bending the thumb) and "B" (bending the index finger) signals for positive and negative z directions, respectively. This interface does not contain Nintendo's proprietary virtual codes. The authors found this to be an awkward way to manipulate depth.

Readers with further interest in these products can contact the manufacturers using the address given in the notes.

REFERENCES

Apple Computer, Inc. (1990). Guide to the Macintosh Family Hardware, Cupertino, CA: Apple Computer, Inc.

Apple Computer, Inc. (1987), HyperCard, (software), Cupertino, CA: Apple Computer, Inc.

Conn, C., Lanier, J., Minsky, M., Fisher, S., and Druin, A. (1989). Virtual Environments and Interactivity: Windows to the Future (panel session), SIGGRAPH '89 Panel Proceedings, 7-18.

Eglowstein, H. (1990). Reach Out and Touch Your Data, BYTE, 15(7), 283-290.

Foley, J.D. (1987). Interfaces for Advanced Computing, Scientific American, October, 257(4), 126-135.

Hyperpress Publishing Corporation, (1988). Icon Factory, (software), Foster City, CA: Hyperpress Publishing Corporation.

Mattel, Inc. (1989). Power Glove Instructions, Hawthorne, CA: Mattel, Inc.

Science Accessories Corporation, (1989). GP8-3D Sonic Digitizer Operator's Manual, Stratford, CN: Science Accessories Corporation.

Symantec Corporation, (1989). Think C User's Manual, Cupertino, CA: Symantec Corporation.

Symantec Corporation, (1989). Think C 4.0 (software), Cupertino, CA: Symantec Corporation.

Turner, C.H. and Green, P. (1987). Human Factors Research on Automobile Secondary Controls: A Literature Review (Technical Report UMTRI-87-20), Ann Arbor, MI: The University of Michigan Transportation Research Institute, October.

Weimer, D. and Gamapathy, S.K. (1989). A Synthetic Environment with Hand Gesturing and Voice Input, CHI'89 Proceedings, May, 235-240.

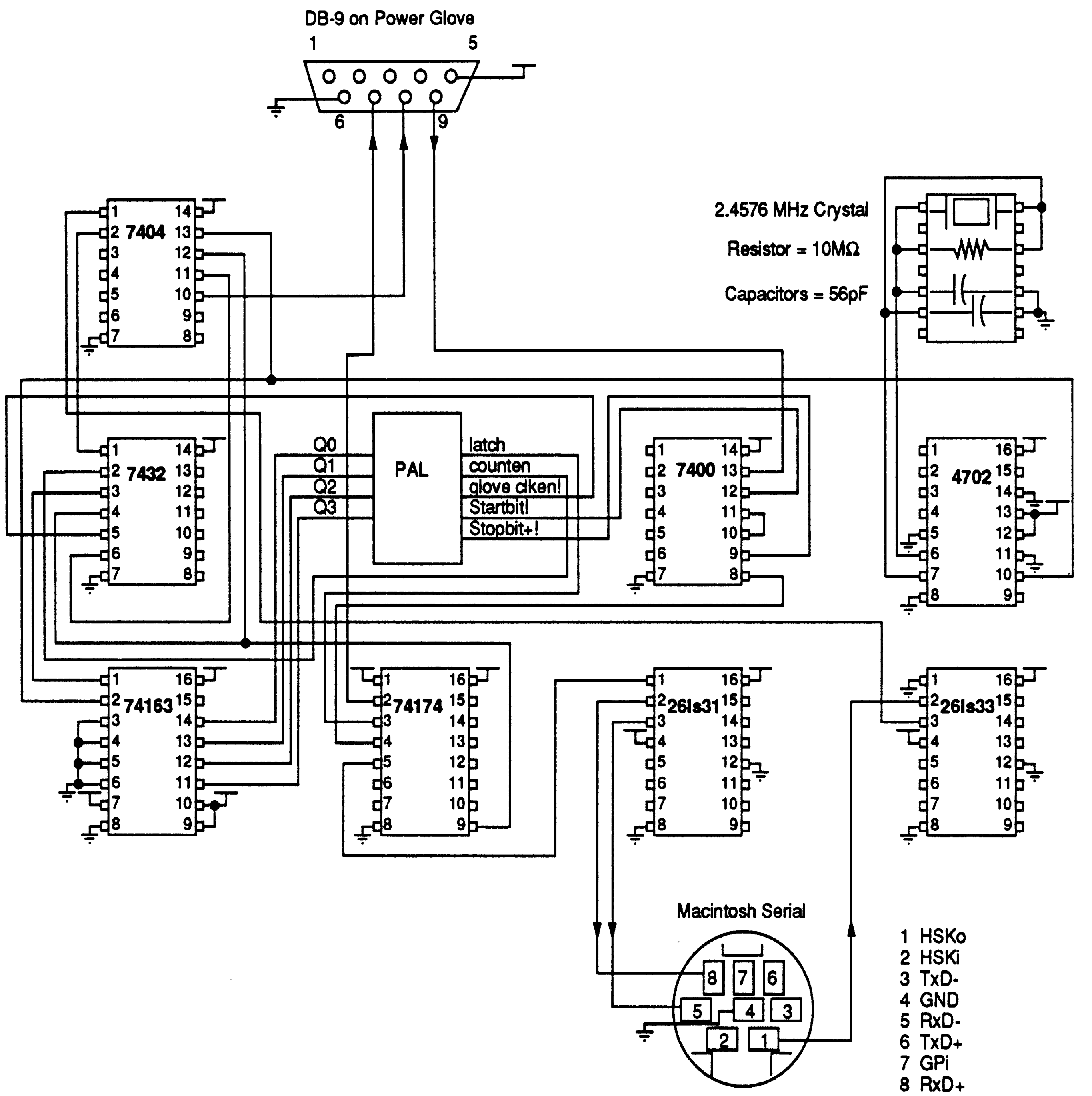
Zimmerman, T.G., Lanier, J., Blanchard, C., Bryson, S., and Harvil, Y. (1987). A Hand Gesture Interface Device, CHI+GI 1987 Conference Proceedings, 189-192.

NOTES

We would like to thank Dave Koziol and John Boreczky for their assistance in developing the software, and George Zachary and Dave Benman of VPL Research, 656 Bair Island Road, Third Floor, Redwood City, CA 94063, for their input.

Transfinite Systems Company, Inc., Box N, MIT Branch P.O., Cambridge, MA 02139.

APPENDIX A
INTERFACE HARDWARE CIRCUIT



APPENDIX B
DATA COLLECTION SOFTWARE

```
/*
Power Glove Data Collection
This program signals the interface box to sample the power glove and reads the resulting data byte.
This program is set up to communicate through the modem port (port A).
This program is terminated with a mouse click.
The timing in this program is for the Macintosh Plus,  adjustments to the timing may be needed for faster machines as
noted in the comments.
*/

#include        <SerialDvr.h>
#include        <stdio.h>
#include        <console.h>
#include        <string.h>

/* Configuration Section !! */

/*      The following 2 defines specify to use the A or Modem port within this program
        to use the printer port, change both occurrences of ".A" to ".B".
*/
#define INPORT          "\p.AIn"
#define OUTPORT         "\p.AOut"

/*      The following define specifies the communications settings to use.  Choose from
        the following list:
                baud300         = 380;      {300 baud}
                baud600         = 189;      {600 baud}
                baud1200        = 94;       {1200 baud}
                baud1800        = 62;       {1800 baud}
                baud2400        = 46;       {2400 baud}
                baud3600        = 30;       {3600 baud}
                baud4800        = 22;       {4800 baud}
                baud7200        = 14;       {7200 baud}
                baud9600        = 10;       {9600 baud}
                baud19200       = 4;        19200 baud}
                baud57600       = 0;        {57600 baud}
        note: the baud rate on the interface hardware must also be altered via the 4702 chip.
        warning: the power glove does not function consistently with a baud rate below 4800.

                stop10          = 16384;    {1 stop bit}
                stop15          = -32768;   {1.5 stop bits}
                stop20          = -16384;   {2 stop bits}
                noParity        = 8192;     {no parity}
                data8           = 3072;     {8 data bits}
*/
#define SERSET          baud9600 + stop10 + noParity + data8

/*      The following define specifies the amount of time in ticks (60 ticks to the
        second) to wait between requesting a byte from the PowerGlove.
*/
#define CHARDELAY     1

/*      The following define specifies the amount of time in ticks (60 ticks to the
        second) that DTR should be asserted to obtain a byte from the PowerGlove.
        It is an integer for loop counts.
*/

#define DEASSERTDELAY        25

/* End of Configuration Section */
/* Convient Defines */
#define         assertDTR       17
#define         negateDTR       18
#define         APPLEID         128
#define         FILEID          129
#define         QUIT            1
#define         ALERT           151

#ifndef         NIL
#define         NIL             0L
#endif

/* Function ProtoTypes */
void            main            (void);
Boolean         initSerial      (void);
char            getChar         (void);
void            alert           (char *);
void            warn            (char *);
void            printChar       (char);
void            doEventLoop     (void);


/* Globals */
int             outRefNum, inRefNum;
FILE            *output;
MenuHandle      menulist[2];
Boolean         hellFrozenOver = FALSE;


void main()
{
        int i;

        InitMac();

        strcpy(console_options.title, "\PDTRTerm");
        fopenc();
        output = fopen("Output", "w");

        doEventLoop();
        fclose(output);
}


InitMac()
{
        InitGraf(&thePort);
        InitWindows();
        InitFonts();
        TEInit();
        InitDialogs(0L);
        InitMenus();
        InitCursor();

        setUpMenus();

        initSerial();

}


setUpMenus()
{
        menulist[0] = GetMenu(APPLEID);
        AddResMenu(menulist[0], 'DRVR');

        menulist[1] = GetMenu(FILEID);
        InsertMenu(menulist[0],0);
        InsertMenu(menulist[1],0);

        DrawMenuBar();
}

Boolean initSerial()
{
        int             result;
        long    count;          /* The number of characters waiting in the serial input buffer */
        char    *theChars;      /* A Pointer used to flush the input buffer */

        result = OpenDriver(OUTPORT,&outRefNum);
        switch (result)
        {
                portInUse:
                        alert("\PPort already in use!");
                        break;
                portNotCf:
                        alert("\PPort not configured for this connection!");
                        break;
                memFullErr:
                        alert("\PNot enough room in the heap zone!");
                        break;
        }
        if (result)
                return(FALSE);

        result = OpenDriver(INPORT,&inRefNum);
        /* We'd probably only get an error in the first case, but copy and paste is easy... */

        result = SerReset(inRefNum, SERSET);
        if (result)
                return(FALSE);

        /* Shouldn't need to do this since we never output anything, but we'll do it anyways. */
        result = SerReset(outRefNum, SERSET);
        if (result)
                return(FALSE);

        SerGetBuf(inRefNum, &count);
        if (count > 0)
        {
                /*      Generally you should use Handles on the Mac to avoid memory fragmentation,
                        but since we're going to free the memory right away we'll use a pointer because
                        it's faster...
                */
                theChars = NewPtr(count);
                FSRead(inRefNum, &count, theChars);
                DisposPtr(theChars);
        }

        return(TRUE);
}


/*      This routine De-asserts DTR for DEASSERTDELAY ticks Asserts it, and then attempts to read a
        character from the serial port.
        Note:  All Control calls should go out the output character channel driver.
        Note:  Delay isn't always exactly accurate (see IM II-384), but it should do for our
        purposes.
*/
char getChar()
{
        long    finalTicks;         /* Total number of ticks from System Startup to end of Delay */
        int     result;             /* These functions return results, but we shouldn't ever get errors */
        long    count;              /* The number of characters waiting in the serial input buffer */
        char    theChar;
        int     i;
        int     watch;


        Control(outRefNum, negateDTR, 0L);

        for (i=0;i<=DEASSERTDELAY;i++);

        Control(outRefNum, assertDTR, 0L);

        count=0;
        watch =0;

        while ((count==0) && (watch<25))
        {
        /* Now we should have a character waitting to be read. */
                SerGetBuf(inRefNum, &count);
                watch++;
        }
        if (count == 0)
        {
                warn("Nada");
                theChar = '\0';
        }
        else
        {
                if (count > 1)
                        warn(">1 : ");

                /* Loop until there are no more characters in the input buffer... */
                while (count != 0)
                {
                        count = 1;          /* We're only set up to read in 1 character! */
                        FSRead(inRefNum, &count, &theChar);
                        printChar(theChar);
                        printf(", ");
                        fprintf(output, ", ");
                        SerGetBuf(inRefNum, &count);
                }
        }
        printf("\n");
        fprintf(output, "\n");
        return(theChar);
}

void printChar(ch)
char    ch;
{

         long    i;

        for (i = 0;i < 8;i++)
        {
                if (BitTst(&ch, i))
                {
```

```
                                printf("1");
                                fprintf(output, "1");
                        }
                        else
                        {
                                printf("0");
                                fprintf(output, "0");
                        }
                }
}

void alert(str)
char *str;
{
  int i;
  ParamText(str,NIL,NIL,NIL);
  i = StopAlert(ALERT,NIL);
}

void warn(message)
char        *message;
{
        printf("%s ", message);
        fprintf(output, "%s ", message);
}

void doEventLoop()
{
        EventRecord     currentEvent;
        Boolean         finished;
        long            time, lastDTR;
        char            ch;

        finished = FALSE;

        lastDTR = TickCount();
        while (!hellFrozenOver)
        {
                time = TickCount();
                if (CHARDELAY + lastDTR < time)
                {
                        ch = getChar();
                        lastDTR = time;
                }

                if (Button())
                {
                        hellFrozenOver = TRUE;
                }
        }
}

doMouseDown(theEvent)
EventRecord             *theEvent;
{
        short           whereDown;
        long            menuData;
        WindowPtr       whichWindow;

        whereDown = FindWindow(theEvent->where, &whichWindow);

        switch (whereDown)
        {
                case 1: /* inMenuBar */
```

```
                    menuData = MenuSelect(theEvent->where);
                    if ((menuData >> 16) == 0)
                            ;                   /* Case of gave up on menu */
                    else
                            doSelection(menuData);

                    if (!hellFrozenOver)
                            HiliteMenu(0);
                    break;

            case 2:  /* inSysWindow */
                    SystemClick(theEvent, whichWindow);
                    break;

            case 3:  /* inContent */
                    SelectWindow(whichWindow);
                    break;
        }       /* end Switch statement */
}

doSelection(menuData)
long    menuData;
{
        short           menuName, menuItem, i;
        char            itemStr[255];
        GrafPtr         savePort;
        Boolean         whome;
        char            ret[255];

        menuName = (menuData >> 16);
        menuItem = (menuData & 0x0000FFFF);

        switch (menuName)
        {
                case APPLEID :
                        GetItem(menulist[0], menuItem, &itemStr[0]);

                        GetPort(&savePort);
                        OpenDeskAcc(&itemStr[0]);
                        SetPort(savePort);
                        break;

                case FILEID     :
                        hellFrozenOver = TRUE;
        }       /* end Switch (menuName) */
}

doKeyDown(theEvent)
EventRecord             *theEvent;
{
        long            menuData;

        if ((theEvent->modifiers & cmdKey) != 0)
        {
                menuData = MenuKey((char)(theEvent->message & 0xFF));
                if ((menuData >> 16) == 0)
                        ;               /* Case of gave up on menu */
                else
                        doSelection(menuData);

                if (!hellFrozenOver)
                        HiliteMenu(0);
        }
}
```

APPENDIX C
HYPERCARD DEMO SOFTWARE

Script for Card BackGround:

```
on doIt samples
 repeat with x = 1 to samples

   put recvChars(1) into askii
   put charToNum(askii) into askii
   put binary(askii) into temp

   --put askii into temp
   put temp into card field tempfield

   if char 1 of temp = 0 then -- hand to right
    if char 3 of temp = 0 then
     -- lower right corner
     set the loc of button showHand to 330,230
    else
     if char 4 of temp = 0 then
      -- upper right corner
      set the loc of button showHand to 330,100
     else
      -- right only
      set the loc of button showHand to 330,166
     end if
    end if
   else
    if char 2 of temp = 0 then -- hand to left
     if char 3 of temp = 0 then
      -- lower left corner
      set the loc of button showHand to 180,230
     else
      if char 4 of temp = 0 then
       -- upper left corner
       set the loc of button showHand to 180,100
      else
       -- left only
       set the loc of button showHand to 180,166
      end if
     end if
    else
     if char 3 of temp = 0 then -- hand down
      set the loc of button showHand to 257,230
     else
      if char 4 of temp = 0 then -- hand up
       set the loc of button showHand to 257,100
      else
       -- middle
       set the loc of button showHand to 257,166
      end if
     end if

    end if
   end if

   -- power glove program 13
   if char 8 of temp = 0 then
    if char 7 of temp = 0 then
     -- All in
     set the icon of button showHand to "handAllIn"
    else
```

```
      -- thumb only in
      set the icon of button showHand to "handThumbIn"
    end if
  else
    if char 7 of temp = 0 then
      -- fingers only in
      set the icon of button showHand to "handFingersIn"
    else
      -- all out
      set the icon of button showHand to "handAllOut"
    end if
  end if

 end repeat

end doIt


function binary fred
 put fred into x
 put (x div 128) into final
 put (x mod 128) into x
 put (x div 64) after final
 put (x mod 64) into x
 put (x div 32) after final
 put (x mod 32) into x
 put (x div 16) after final
 put (x mod 16) into x
 put (x div 8) after final
 put (x mod 8) into x
 put (x div 4) after final
 put (x mod 4) into x
 put (x div 2) after final
 put (x mod 2) after final
 return final
end binary
```