Trackball-Interfacing Techniques for Microprocessors

This interfacing approach lets you adapt trackball devices to your interactive personal computer applications

by Edward W. Andrews

The age of interactive computing is upon us, and a variety of I/O (input/output) devices have been developed to supplement the keyboard in providing convenient human-to-machine interfacing. Unfortunately, in the past only simple joysticks and pushbuttons have been priced low enough to suit home computer applications. The LT200, a recent product from Disc Instruments of Costa Mesa, California, has brought the trackball within reach of the home computer market. This product provides accurate positioning of the cursor displayed on a CRT (cathode-ray tube), and it provides TTL-compatible outputs that can be readily interfaced with a microcomputer without elaborate and costly circuitry. Moreover, the LT200 costs less than $100, representing a major savings compared to the trackballs designed for commercial and military markets that cost more than $1000.

This article describes general trackball-interfacing techniques and a simple hardware/software interface approach that can be adapted easily to any home system.

Trackball Concepts

A trackball is an interactive control that consists of a solid ball, 1½ to 3 inches in diameter, mounted in a base such that part of the ball's surface is exposed, allowing the ball to be rolled with the palm or fingertips. The ball's rolling motion is coupled to optical encoders that generate pulses to indicate the direction and rate of ball rotation. These pulses can be coupled to a microprocessor, which can then create proportional X-Y motion of a CRT-displayed cursor.

A simple hardware/software-based trackball-interfacing approach can be adapted to any personal computer system.

The precision with which a trackball can be moved suits it to detailed interactive graphics applications. Commercially it has seen wide use in computer-aided-design (CAD) systems. Other applications include interactive analysis of medical X-ray, nuclear, and ultrasound images. Some video-arcade games have also featured trackball controls.

Figure 1 shows the internal construction of the LT200 trackball. Two optical interrupter disks (the slotted wheels) form the basis of an optical encoder for each trackball axis. The ball rests on two perpendicular rods on which the interrupter disks are mounted. Any movement of the ball causes at least one of the rods to rotate, in turn causing disk rotation and, thus, pulse-train generation. The frequency of these pulse-train signals is proportional to the ball-rotation speed along each axis. Most trackballs generate 200 to 500 pulses per revolution. The LT200, for example, generates 480 pulses per revolution.

Because the two rods are perpendicular, one encoder's output represents X-axis movement while the other encoder's output represents Y-axis movement. Each encoder includes sensing logic that determines forward or reverse (or left or right) movement along the corresponding axis; a trackball device therefore furnishes four signals indicating movement in +X, −X, +Y, and −Y directions. When used to control cursor positioning on a CRT, the trackball must be properly oriented with

respect to the CRT to obtain the proper correlation between ball movement and cursor movement.

Basic Interfacing Concepts

Note that a trackball is essentially an incremental or relative input device. By rolling the trackball, an operator signals his intent for the displayed cursor symbol to move in a given direction and at a given rate, away from the current cursor position. In response to trackball rotation, the cursor moves to a new position based on its current position.

A review of some fundamental CRT display-addressing concepts illustrates how the incremental trackball signals can be interfaced to a computer. Consider a CRT display having an X-Y matrix of 256 by 256 pixels (picture elements). Within such a matrix, the displayed cursor location on the CRT can be specified by an X, Y number pair. One 8-bit number can uniquely define all possible horizontal X pixel locations, or addresses, on the CRT display, starting with 0 on the left edge and extending to 255 at the right edge of the display. Similarly, a second 8-bit value uniquely defines all possible vertical Y pixel locations of the CRT, starting with 0 at the bottom and ending at 255 at the top edge of the display. If the X and Y values defining the absolute cursor location can be varied in response to the trackball, an interactive control results.

Implementation Approaches

As figure 2 shows, a trackball interface can be built using two simple hardware up-down counter circuits.

A trackball is essentially an incremental or relative input device.

Here the X-axis uses one counter and the Y-axis uses a second. With the trackball +X output connected to the up-count clock, and the −X trackball output connected to the down-count clock, the counter increments and decrements appropriately in response to trackball X-axis movement. The Y-axis trackball outputs are connected similarly to a second counter, which in turn responds to Y-axis trackball movement. These counter circuits can then be interfaced to a microprocessor's data bus through an input port, allowing the processor to read or periodically poll the input port to determine the current absolute X, Y trackball coordinate. Using this data, the microprocessor can position the cursor on the CRT. If this input port is read and the displayed cursor position is updated at a high rate (more than 25 times per second), an interactive control results.

As shown, upper- and lower-limit detection logic must be included in the counter design to prevent counter roll-over, which could occur if the trackball is unceasingly rolled in one direction. The X-axis counter, for example, must be inhibited from further up-counting when the rightmost pixel coordinate, 255, is reached. Should such limiting be omitted, further +X pulses from the trackball would cause the counter to overflow from 255 to 0. If this overflow were to occur, you would see the cursor jump abruptly from the far right side of the display screen (X address = 255) to the far left side of the display screen (X address = 0). With boundary limiting properly implemented, the trackball appears to slip whenever a display screen edge is encountered. Such boundary limiting must also be included for the left, top, and bottom cursor boundaries.

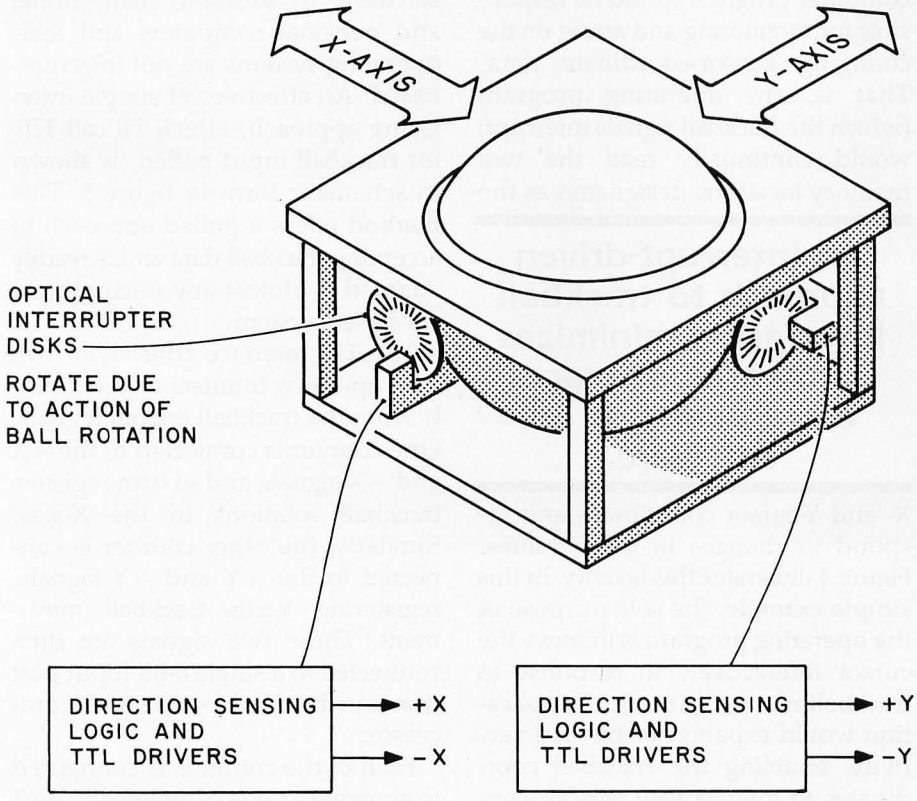


Figure 1: *An internal look at Disc Instruments' LT200 trackball.*

An Interrupt-Based Interface

We have seen the basic trackball interfacing concepts demonstrated with a hardware-intensive approach. While counters can be readily configured to directly implement a trackball interface, other less hardware-intensive approaches are also possible. One such method uses interrupt concepts and software-based up-down counters to respond to the trackball output pulses. In other words, the counters just detailed are functionally implemented in software, and the trackball output pulses are connected as vectored interrupts to a microprocessor system.

With an interrupt-based approach, you must think of the current trackball X- and Y-cursor coordinates as residing in two locations in the micro-

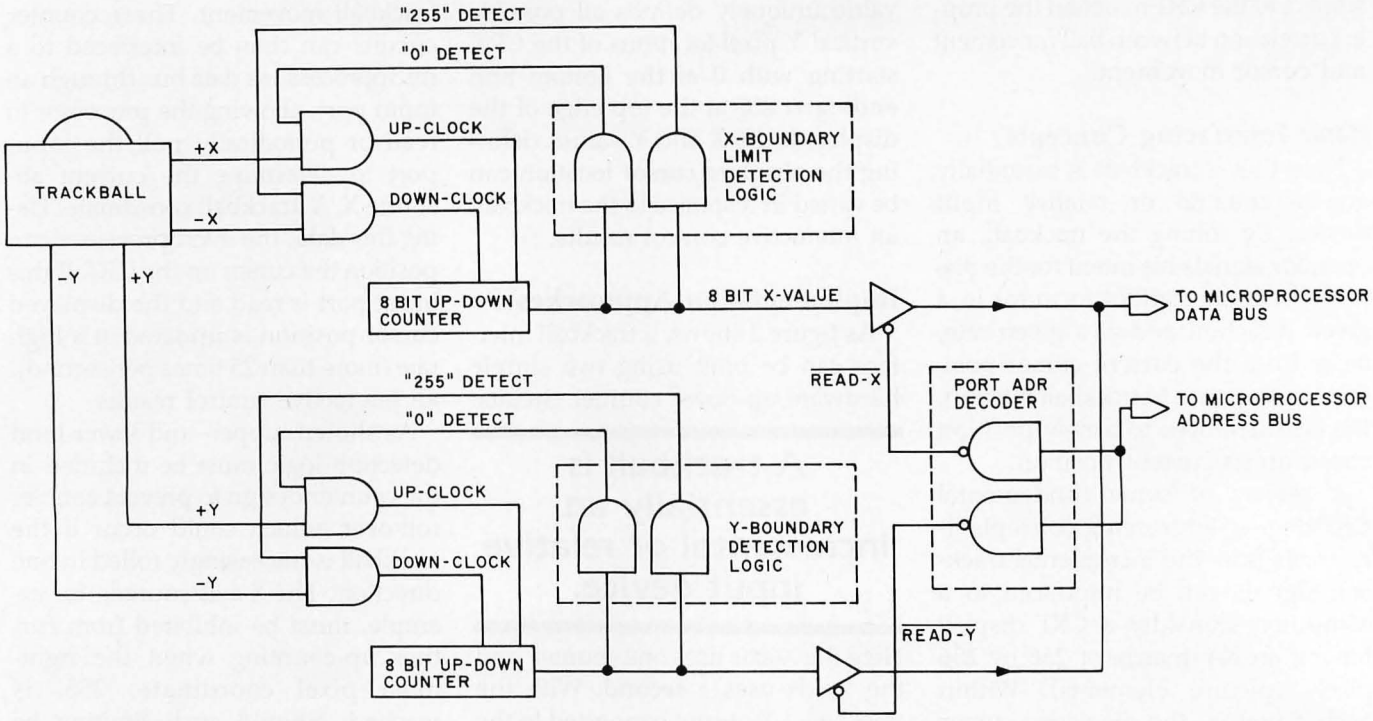


Figure 2: *A block diagram of a fundamental hardware-based trackball interface.*

processor's RAM (random-access read/write memory). Rather than drive a hardware-counter circuit, the trackball signals are now connected as vectored interrupts to the processor system. An interrupt controller, such as an Intel 8259, can be used to add vectored-interrupt capability to a computer system. Alternately, the trackball can be interfaced to an existing interrupt structure. In any event, with an interrupt-based approach, the software interrupt-handler routines operate as up-down, software-based counters.

With the trackball signals connected as interrupts, for example, as a +X trackball pulse is received, the microprocessor is vectored to an interrupt-service routine whose job is to increment the data contained in the X-cursor-coordinate memory location. Similarly, the −X interrupt-service routine decrements the data contained in the X-cursor-coordinate memory location in response to a −X trackball pulse.

Figure 3 shows simple flowcharts of the four interrupt-service routines required for a fully interrupt-based implementation. Notice that upper/lower boundary detection and limiting is also included in these interrupt routines.

With the interrupt routines maintaining and updating the absolute cursor X, Y coordinates, the operating computer program would be responsible for monitoring and acting on the changing cursor-coordinate data. That is, the operating program (which the trackball signals interrupt) would continually read the two memory locations designated as the X- and Y-cursor coordinates and respond to changes in these values. Figure 4 illustrates this activity. In this simple example, the sole purpose of the operating program is to move the cursor interactively in response to trackball motion. An actual application would expand on this basic activity, enabling the trackball coordinates to interactively vary or control higher-level functions.

An interrupt-driven approach to trackball interfacing minimizes hardware but places extra demands on software.

The TIP Approach

Although an interrupt-driven approach to trackball interfacing can minimize the interface hardware, it does place extra demands on the software. In addition, many home and personal computers and their operating systems are not interrupt-based. An effective yet simple interfacing approach, which I'll call TIP, for trackball input polled, is shown in schematic form in figure 5. This method offers a polled approach to accepting trackball data and is readily adapted to almost any microprocessor-based system.

The TIP interface consists of two 4-bit up-down counters driven directly from the trackball output signals. One counter is connected to the +X and −X signals and in turn registers trackball rotations in the X-axis. Similarly, the other counter is connected to the +Y and −Y signals, registering Y-axis trackball movement. These two signals are then connected to a single 8-bit input port that can be read by the microprocessor.

Each of the counters is configured to generate a two's complement, 4-bit signed number. As the TIP input port

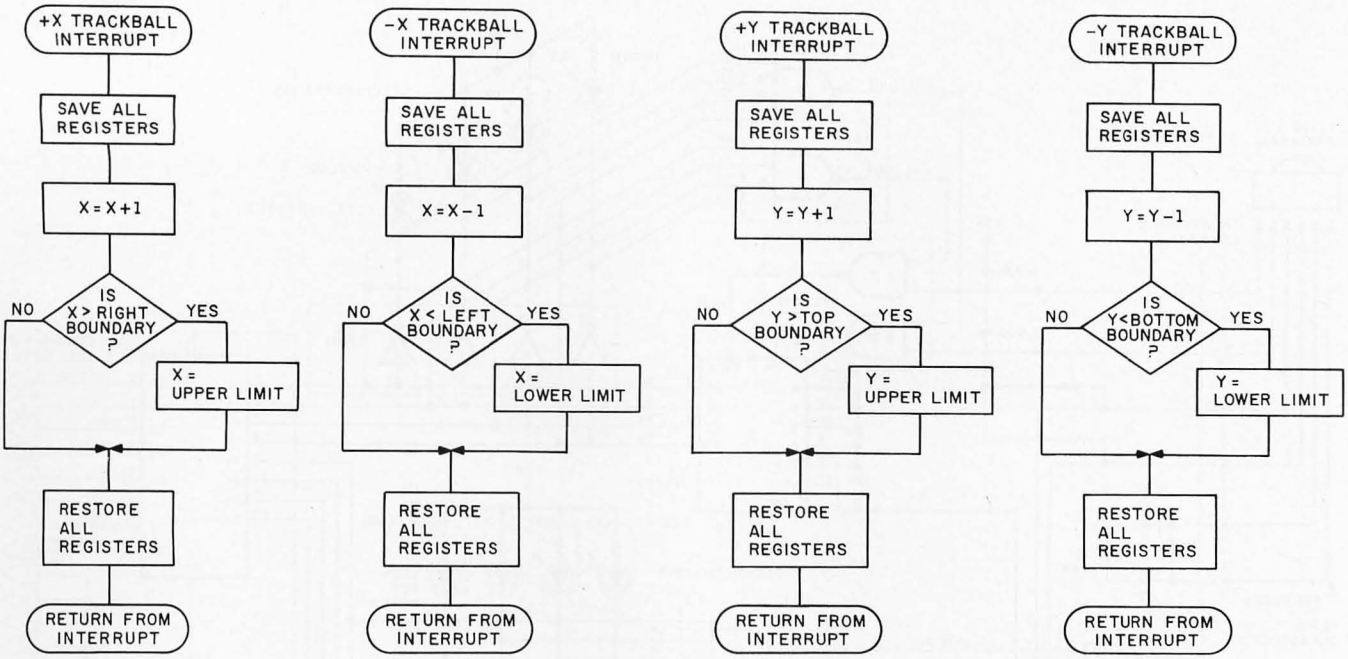


Figure 3: *Flowcharts for the four interrupt-handler routines required for a vectored-interrupt trackball-interface approach. Note that boundary checking is included in these routines.*

is read by the microprocessor, the two numbers retrieved represent an accumulation of the operator's most recent trackball actuation. In effect, the data read is an X, Y vector, indicating the direction and magnitude (speed) of trackball rotation. The software then alters the current cursor-location values based on this move vector. This method differs from the approach suggested by figure 2, in which the interface-counter hardware actually holds the absolute cursor location, rather than a relative move count. Figure 6 shows the data format as it is read from the TIP input port.

TIP Circuit Details

The four trackball signals, +X, −X, +Y, and −Y, are received and gated by the Schmitt-trigger device IC1 in figure 5. We chose a Schmitt device to increase the noise immunity of the input circuitry, therefore reducing the chance of random electrical noise from adversely affecting the counter operation. IC4 (X-axis) and IC5 (Y-axis) are TTL-type 74LS192 up-down, 4-bit binary counters. These counters are structured to count symmetrically from zero, each able to count within the two's complement number range from −7 to +7. NAND gates IC2b (X-axis) and IC3b (Y-axis) detect the uppermost count boundary, +7, and work in conjunction with IC1 to prevent an ongoing stream of up-count clocks (+X and +Y trackball signals) from causing an undesired counter overflow. Similarly, NAND gates IC2a (X-axis) and IC3a (Y-axis), detect the lowermost count boundary, −7, and, with IC1, prevent an ongoing stream of down-count clocks (−X and −Y trackball signals) from causing an undesired counter underflow.

IC7 is an 8-bit clocked register with three output states and is used as a combination data-latch and data-bus driver. A latch here ensures that a stable, unchanging data byte would be presented during the processor's READ operation. In addition, just after the data latch is clocked to hold the current trackball X, Y data pair, both X and Y counters are reset to zero. Thus, after every TIP register is read, the counters start over in accumulating operator input.

The LED (light-emitting diode) indicators are optional and not required for circuit operation; however, I found them invaluable in debugging my wire-wrapped prototype.

Address and Data-Bus Interface

Comparators IC8 and IC9 form the address-bus decoder. These two

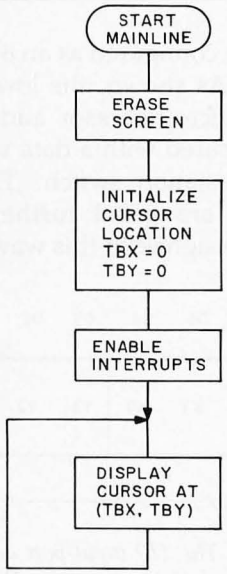


Figure 4: *The flowchart of a simple main program designed to work with the figure 3 interrupt routines. The trackball-coordinate values TBX and TBY are the variables that the interrupt handlers update.*

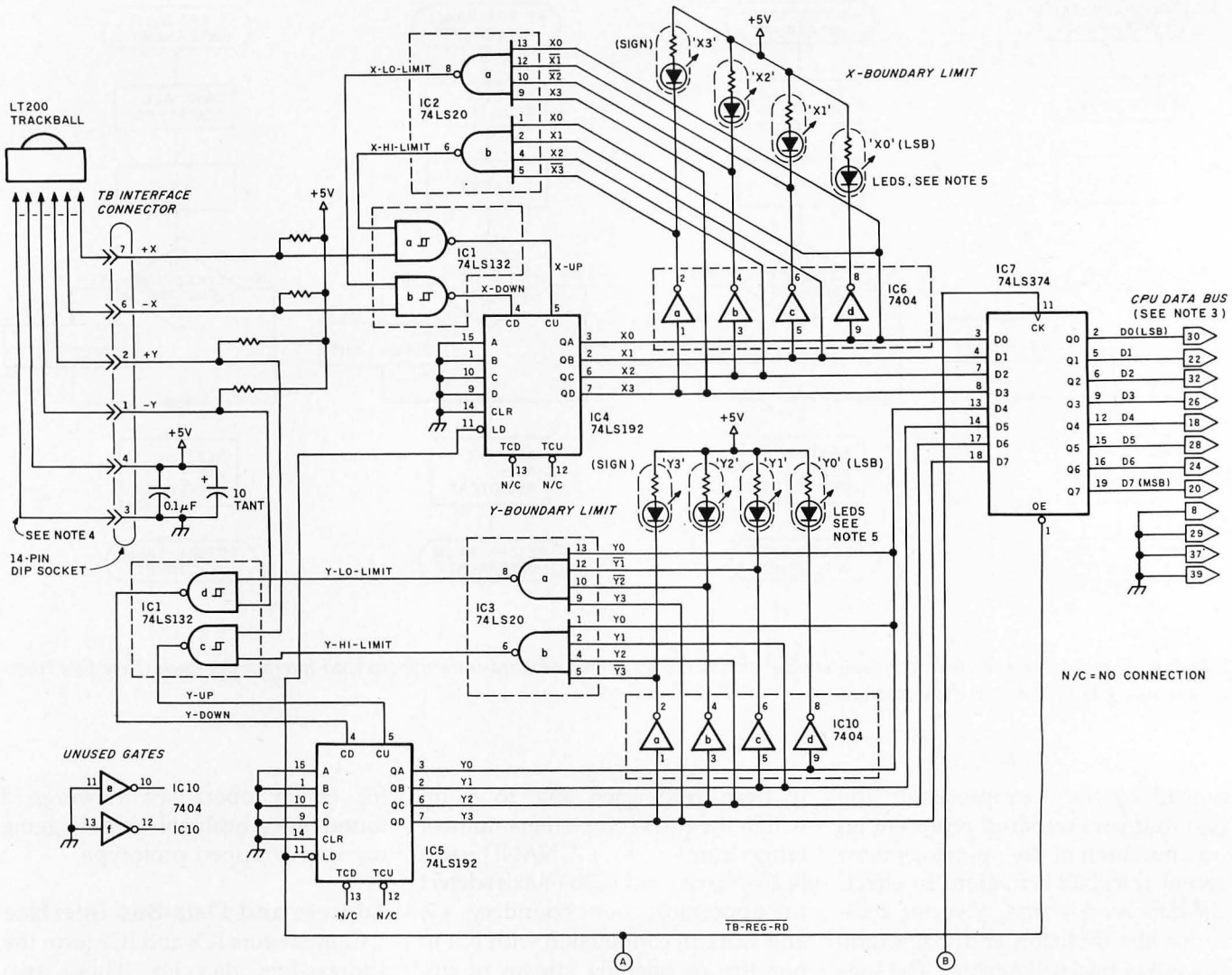


Figure 5: *The TIP (trackball input polled) interface schematic. Connector pinouts are shown for a TRS-80 Model I computer.*

chips are configured as an 8-bit comparator. As shown, the lower 8 bits of the microprocessor address bus are compared with a data value, set by an 8-station switch. The comparators are gated further by an IOREAD signal. In this way, the TIP input register is mapped into the microprocessor I/O space. The actual port assignment is determined by the switch settings of S0 (LSB) through S7 (MSB). When a given switch position is open (off), a logic 1 is set; closing the switch (on) results in a logic 0 state.

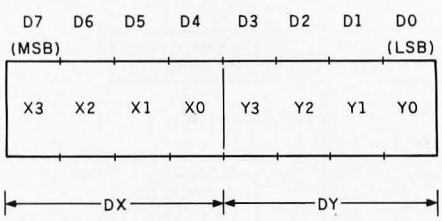


Figure 6: *The TIP input-port data format. The lower 4 bits represent a delta Y (DY) value, and the upper 4 bits represent a delta X (DX) value. This pair of two's complement values, each having a number range from –7 to +7, represents the most recent operator actuation of the trackball.*

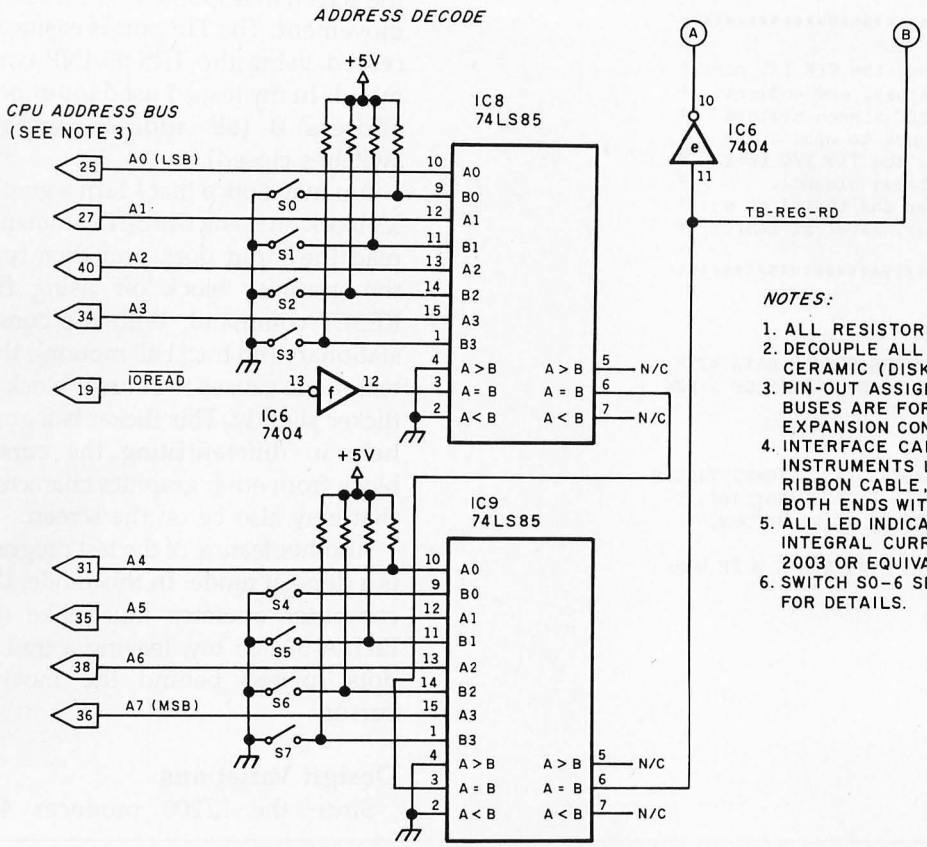
Although a TRS-80 computer was used to demonstrate concept feasibility, the corresponding address bus, data bus, and IOREAD signals of any microcomputer can be connected to the TIP interface. If desired, the TIP data port can even be memory-mapped. For this memory-mapped approach, the address-decoding circuitry has to be expanded to compare 16 bits (or more). Additional 74LS85 comparator chips can be cascaded, or other combinational logic techniques can be used. Any high-true, address-decoding signal can be used to drive the point called "TB-REG-RD" of figure 5; however, TB-REG-RD must occur at least one gate delay before TB-REG-RD.

As shown, the microprocessor data bus is connected to the output of IC7. Data bit D7 is the most significant bit (MSB), and D0 is the least significant bit (LSB).

A TRS-80 Demonstration

The address and data bus pinouts shown in figure 5 correspond to the expansion-interface pin assignments for the TRS-80 Model I computer, which I used to evaluate and test the circuit concepts. This expansion interface connector makes available all the key Z80 processor signals needed to connect the TIP circuitry.

Listings 1 and 2 show simple



NOTES:

1. ALL RESISTORS ARE 2.2 KΩ, 0.25 W, UNLESS NOTED.
2. DECOUPLE ALL CHIPS WITH 0.1 μF, 50 V MONOLITHIC OR CERAMIC (DISK) CAPACITOR.
3. PIN-OUT ASSIGNMENTS SHOWN FOR ADDRESS AND DATA BUSES ARE FOR CONNECTION TO A TRS-80 MODEL I EXPANSION CONNECTOR CABLE.
4. INTERFACE CABLE BETWEEN TIP CIRCUIT AND DISK INSTRUMENTS LT200 SHOULD BE 14-CONDUCTOR RIBBON CABLE, LENGTH AS REQUIRED, TERMINATED ON BOTH ENDS WITH A MALE, 14-PIN DIP PLUG.
5. ALL LED INDICATORS SHOULD BE PC-MOUNTABLE; WITH INTEGRAL CURRENT LIMIT RESISTORS (DIALIGHT 547-2003 OR EQUIVALENT).
6. SWITCH S0-6 SETS I/O PORT ASSIGNMENT. SEE TEXT FOR DETAILS.

Listing 1: *A TRS-80 Model I (Level II BASIC) main program that controls cursor movement in response to trackball actuation.*

```
10    REM ******************************************
15    REM *                                        *
20    REM *       TIP DEMONSTRATION PROGRAM        *
30    REM *        ------------------------        *
40    REM * This program will move a block cursor  *
50    REM * about the CRT screen in response to    *
60    REM * the movement of the trackball.  A      *
70    REM * mode can be selected which will result *
80    REM * in "Etch-A-Sketch" like operation.     *
90    REM * This program was written and tested on *
100   REM * a TRS-80, Model I, computer, with      *
110   REM * Level II BASIC.  Note. the TRS-80,     *
120   REM * Model I computer has a CRT screen      *
130   REM * resolution of only 128 X 48 PIXELS.    *
140   REM ******************************************
150   REM
160   CLS
170   REM --- SET INITIAL X,Y VALUES
180   Y=20:X=50
190   REM --- ASK OPERATOR IF HE WANTS TO TRACE
200   PRINT "ENTER 1 FOR TRACE MODE, 0 FOR NO TRACE"
210   INPUT A
215   CLS
220   REM --- MAIN LOOP STARTS HERE----------------
230   REM --- TURN CURSOR BLOCK "ON"
250   SET(X,Y)
260   REM --- READ TIP I/O PORT AND GET DX, DY
270   GOSUB 800
280   REM --- TURN CURSOR BLOCK "OFF"
290   REM     THIS ON-OFF SEQUENCE CAUSES
300   REM     CURSOR TO FLICKER SLIGHTLY WHICH
310   REM     DIFFERENTIATES IT FROM OTHER
320   REM     ON-SCREEN GRAPHICS BLOCKS
330   RESET(X,Y)
340   REM --- WAS TRACKBALL ROLLED? IF NO, LOOP BACK
350   IF DX=0 AND DY=0 GOTO 250
360   REM --- YES, TB WAS ROLLED; CHECK IF TRACE SELECTED
370   IF A=0 GOTO 410
380   REM --- IF TRACE SELECTED, TURN CURRENT BLOCK ON
390   SET(X,Y)
400   REM --- NOW UPDATE CURSOR COORDINATES
410   X=X+DX
420   REM --- NOTE: SUBTRACT DY BECAUSE "+Y"
430   REM     DIRECTION IS "DOWN" ON TRS-80 SCREEN
440   Y=Y-DY
450   REM --- PERFORM BOUNDARY CHECKING AND LIMITING
460   IF X>127 X=127
470   IF X<0   X=0
480   IF Y>47  Y=47
490   IF Y<0   Y=0
500   GOTO 250
```

Listing 2: *A subroutine that reads the TIP port and separates the DX and DY elements.*

```
700     REM ***********************************************
710     REM *                                             *
720     REM * This subroutine will read the TIP I/O port, *
730     REM * separate the X and Y values, and convert    *
740     REM * them into a pair of BASIC signed numbers    *
750     REM * which the mainline can use to update the    *
760     REM * cursor position.  Here, the TIP I/O port    *
770     REM * was 0 (All address switches closed).        *
780     REM * This routine was written and tested on a    *
790     REM * TRS-80, Model I computer, level II BASIC    *
800     REM *                                             *
810     REM ***********************************************
850     REM
860     REM --- READ THE I/O PORT
870     TIP=INP(0)
880     REM --- SEPARATE X AND Y FROM THE SINGLE DATA BYTE
890     REM     AND PLACE EACH IN THE LOWER 4-BITS OF A NEW
900     REM     VARIABLE PAIR, DX AND DY
910     DX=(TIP AND 240) * .0625
920     DY=(TIP AND 15)
930     REM --- NOW CONVERT THESE NUMBERS INTO SIGNED VALUES
940     REM     WHICH TRS-80 BASIC WILL UNDERSTAND! 1ST, X
950     REM --- IF THE 2'S COMP NUMBER IS +, WE'RE DONE
960     IF DX < 8 GOTO 1000
970     REM --- IF THE 2'S COMP NO. IS -, THERE'S MORE WORK
980     DX = DX - 16
990     REM --- NOW DO THE Y VALUES
1000    IF DY < 8 GOTO 1030
1010    DY = DY - 16
1020    REM --- ALL DONE.....
1030    RETURN
```

TRS-80 Level II BASIC main and subprograms that read the TIP input port and move a graphics block around the screen in response to the trackball movement. The TIP port is easily accessed using the TRS-80 INP command. In my tests, I used input port address 0 (all address-compare switches closed).

You may notice that I turn a graphics block on using the SET command, read the input port, and then turn the graphics block off using the RESET command. With the cursor stationary (no trackball motion), this technique causes the cursor block to flicker slightly. This flicker is a great help in differentiating the cursor block from other graphics characters that may also be on the screen.

Another feature of the test program is a deposit mode. In this mode, the computer operates much like the Etch-A-Sketch toy, leaving a trail of dots (pixels) behind the moving cursor.

Design Variations

Since the LT200 produces 480

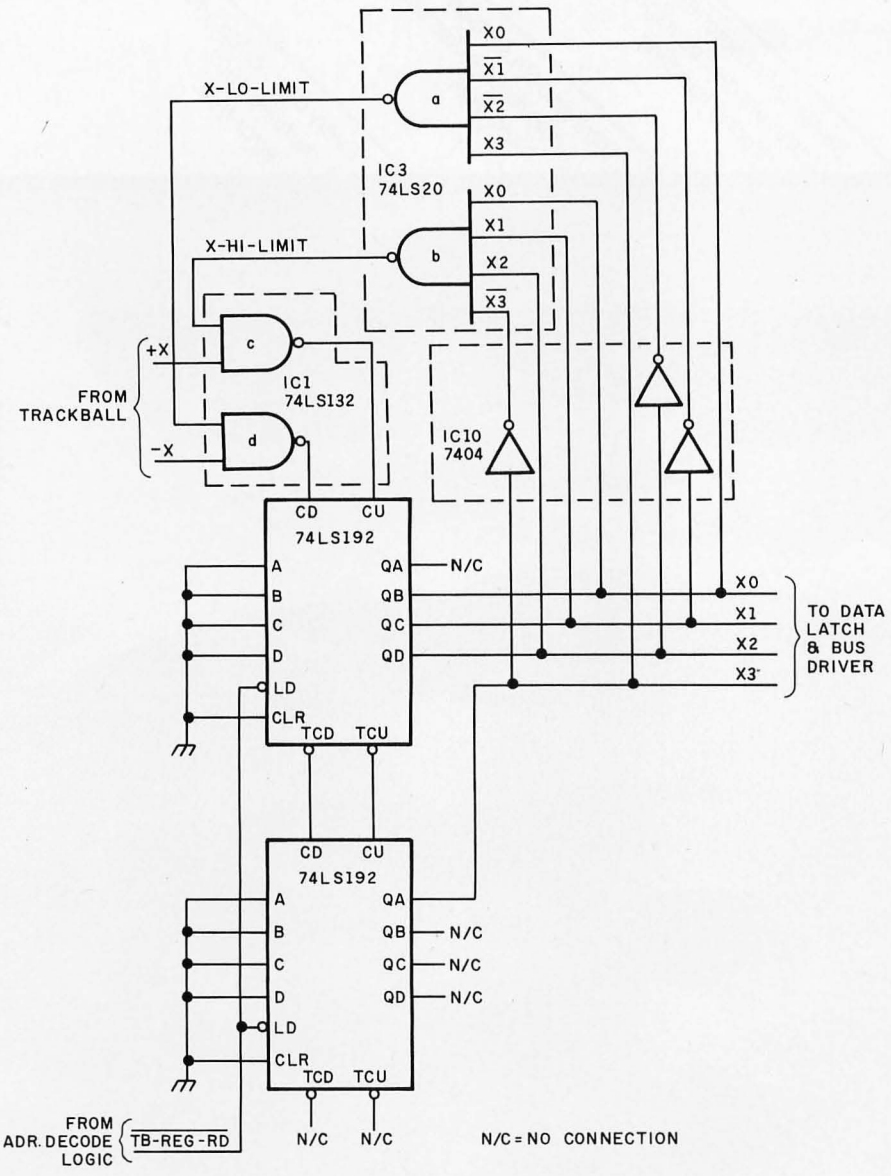


Figure 7: *A design variation that allows the TIP trackball counters to prescale the trackball pulses.*

pulses per revolution, just over half a rotation of the LT200 trackball element moves a displayed cursor from one screen edge to the other screen edge when operating within a 256- by 256-pixel matrix. For detailed cursor placement, at which a trackball device intrinsically excels, you may find the trackball excessively sensitive with the LT200/TIP interface scheme as shown. There are two ways to reduce the trackball sensitivity: a simple hardware change or a software approach.

To reduce the trackball sensitivity, it is necessary to prescale the incoming trackball pulses. Figure 7 shows how an additional 74LS192 counter circuit can be connected to each axis of the TIP interface. This approach extends the effective range of the counter from 4 bits to 8 bits. The TIP interface port and the upper/lower boundary checking, however, are still connected to only 4 bits. As shown, the 4 bits selected ignore the LSB of the first counter chip, effectively dividing by two the trackball pulses seen by the TIP I/O port, resulting in a trackball that produces only 240 pulses per revolution. If desired, the two lowest order bits of the first counter can be ignored, resulting in a trackball that is prescaled by 4, effectively producing only 120 pulses per revolution.

These hardware variations can be made to the design as shown by the TIP schematic of figure 5. To maintain symmetry, both the X and Y axes should be treated in the same fashion.

The scaling operations performed by the hardware changes of figure 7 can also be performed in software after the TIP data port is read. A simple divide-by-two operation can be applied to the X and Y data values. However, the effective number range that the TIP X and Y trackball values can represent is reduced from the −7 to +7 range to a −3 to +3 range. This reduced range adversely affects the ability of the trackball interface to represent a vector-like value indicating the direction and magnitude of the desired cursor move. The hardware approach as discussed, however, prescales the trackball pulses while completely retaining this vector-like characteristic.

Summary

A trackball can be interfaced easily to a personal computer. The TIP approach is both simple and inexpensive. A TRS-80 demonstration validated the design approach. While the TRS-80 did prove effective for evaluating the concept, I feel that a trackball device is better suited to higher resolution graphics systems having a minimum of a 150- by 150-pixel matrix.

Overall, the Disc Instruments LT200 trackball is well constructed. The design is mechanically simple, having few moving parts. While perhaps a bit lightweight to survive the demands of a video arcade, it is well suited to the home and office environment.

With the advent of the low-cost LT200 and similar products that no doubt will soon be offered by other vendors, the time has come to add the power and convenience of the trackball to the home computer.■

Edward W. Andrews (18640 Arden Ave., Brookfield, WI 53005) holds a B.S. degree in computer science and works at General Electric's Medical Systems Operation. In his spare time, he enjoys volleyball, racquetball, and home computing with emphasis on computer-graphics applications.