Foreword

This is an article which appeared in Electronic Design for Sept. 17, 1992, a trade journal for exchanging engineering ideas and techniques, and is reprinted by gracious permission from Endeavor Business Media. In this article, the author James Kuzdrall explains well what happens in a lightly damped system to create errors in reading encoders (this project is very susceptible to these errors). He presents two approaches to accommodating the conditions and correctly interpreting each quadrature transition. The polling approach's code is written in assembly language, and I have translated it into Arduino C and added commenting to help explain the operation.
Chris Howie

# BUILD AN ERROR-FREE ENCODER INTERFACE

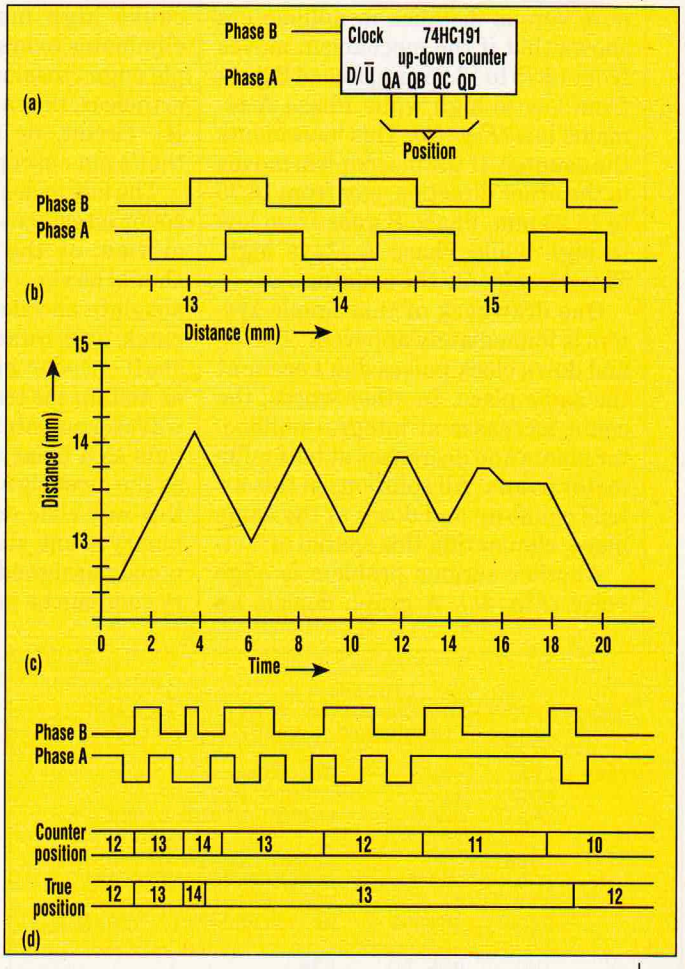
## THREE INCREMENTAL-ENCODER INTERFACE DESIGNS WARD OFF BUGS CAUSED BY LIGHTLY DAMPED MECHANISMS.

Inexpensive incremental optical encoders offer a simple way to monitor changes in mechanical position. Although connecting control electronics to the two-wire output seems straightforward at first, lightly damped mechanisms can befuddle a simple interface. There are ways, however, to develop interfaces that operate error-free with any mechanism, regardless of damping. This article will discuss three designs ranging from purely hardware to purely software approaches. In addition, these interfaces can increase the encoder's resolution by two or four times in many applications.

Conceptually, an encoder consists of a light source that shines through a pattern of opaque and clear bars to reach a photodetector. Placing a bar pattern at the edge of a shaft-mounted disk detects angular changes, whereas placing a linear bar pattern on a strip detects translation. The shadow and light image produces logic-level transitions as the pattern moves through the optical path. Several such patterns and paths can produce a binary code, giving the current position directly. These non-incremental encoders, called absolute encoders, require one path for each bit, which makes them complex at even moderate resolutions. In contrast, the incremental encoder uses only two optical paths. The first gives a simple on or off at the desired resolution. The second path has the same resolution, but is mechanically offset by a quarter period to produce a quadrature signal.

The incremental encoder usually measures displacement relative to a known position. Therefore, when the circuit is turned on, the servomechanism drives to a reference position. Mechanical switches or photo-interrupters often provide this independent reference. Once the reference position is

JAMES A. KUZDRALL
Intrel Service Co., Box 1247,
Nashua, N.H. 03061; (603) 883-4815.




**1. A BINARY UP-DOWN COUNTER implements a simple encoder interface (a). Graphing encoder outputs versus distance shows how the movement of Phase A and Phase B increment or decrement the counter (b). A second graph showing distance plotted against time for an under-damped servo illustrates the problems of edge noise (c). After the servo's motion stabilizes, the counter reading is off by two from the actual position (d).**

found, the circuit measures absolute displacement by counting pulses from the incremental encoder. Quadrature phasing enables the circuit to deduce the travel direction and determine whether the count should be added to or subtracted from the total.

Phase A of the incremental encoder could be designated as the direction and Phase B as the clock *(Fig. 1a)*. Reversing this assignment merely changes the travel direction that causes the count to increase. A circuit built with the 74HC191 binary up-down counter as a cascadable count accumulator will implement this concept.

A graph that plots the encoder outputs versus distance in millimeters shows that if the mechanism moves from 12.75 to 13.25 mm, Phase B goes from low to high while Phase A remains low *(Fig. 1b)*. This increments the counter. If the mechanism moves in the other direction, say from 13.75 to 13.25 mm, Phase B goes from low to high while Phase A stays high. This decrements the counter.

One drawback of this simple system is immediately apparent: The up and down clock pulses don't occur at the same place. In other words, the count increases at integral millimeter points and decreases at half millimeter points. All solutions in this article count up and down at the same place, eliminating this spatial offset.

Another serious problem is edge noise *(Fig. 1c)*. A graph depicts an under-damped servo that moves from 12.70 to 13.70 mm, overshoots, rings for a few cycles, comes to rest, and then moves back to its original position. Waveforms trace the phase line levels and the 74HC191 count as this motion progresses *(Fig. 1d)*. The waveforms show that, after the movement from 12.7 to 13.7 mm, and then back to 12.7 mm, the counter reading is off by two from the true position.

This error source is overlooked (or wisely ignored) in many systems because high mechanical damping eliminates overshoot, making multiple edge crossings improbable. Nevertheless, the safest solution uses an IC, circuit, or software algorithm that's not susceptible to this error.

The key to designing error-free encoder interfaces lies in the conceptual view of the output lines. They should be viewed as bit lines of a 2-bit absolute encoder rather than as the clock and quadrature. A two-bit shaft encoder will rotate many times as distance is traversed *(Fig. 2)*. The waveforms interpret the phase outputs as a binary code, with Phase B as the most significant bit. Although this raw code doesn't progress as a binary count, states can be unambiguously assigned. If necessary, binary code can be produced by inverting Phase A when Phase B is high. Clock-pulse-offset and edge-noise errors are eliminated if the count increments when state 3 changes to state 0 and decrements when state 0 changes to state 3.

One hardware solution relies on a propagation-delay-reset to produce a narrow clock pulse for the up-down counter *(Fig. 3)*. Both flip-flops are usually in the set condition, holding the counter's clock inputs high. The output can't be clocked to low unless Phase A at the flip-flop's D input is low. Phase B clocks both flip-flops—U2A on rising edges and U2B on falling edges.

**BOOLEAN LOGIC FOR ENCODER INTERFACE**

| | | | | | |
|---|---|---|---|---|---|
| Phase B | 0 | 1 | 0 | 1 | (before change) |
| Phase A | 0 | 0 | 1 | 1 | |
| Count Up | 0 | 1 | 1 | 0 | (desired operation) |

Only U1D needs to be an exclusive-OR gate. The DIP's remaining three gates are gainfully employed by serving other logic functions. Wiring one input high makes the other exclusive-OR input invert. Conversely, a permanent low on one input makes the other input noninverting. The spare gates function as either inverters or delay buffers, as needed.

If Phase A is high when Phase B switches, the flip-flop clocks to the set condition. Thus, the output stays the same. Referring to the 2-bit absolute encoder of Figure 2, this transition occurs between states 1 and 2 (either direction), and count shouldn't change.

If Phase A is low when Phase B changes, the flip-flop clocks to the clear condition and the output goes low. The typical exclusive-OR gate (U1A or U1C), propagates a low to the Set input after an internal delay of about 10 ns. In turn, the Set input brings the output back to high after a propagation delay of about 14 ns. The output stays low for 10 + 14 = 24 ns, long enough to exceed the 74HC193's minimum clock width of 22 ns. Using the CMOS MM74C74/MM74C86 combination (pinouts differ) increases the pulse width to about 240 ns.

The circuit produces two additional bits of binary resolution using the 2-bit absolute encoder concept. These outputs would increase the precision (but not necessarily the accuracy) of a 5 counts/mm linear encoder to 20 counts/mm. A circuit configured from the same parts can produce clock and direction outputs for counters like the 74HC193. This slightly simpler form requires four exclusive-OR gates and one flip-flop.

An interrupt-driver scheme replaces the counter ICs of Figure 3 with registers in the microprocessor *(Fig. 4)*. This interface uses the 6805's external interrupt input (IRQ) and the bit 0 and 1 lines from input/output (I/O) port C. When $PC_0$'s out-

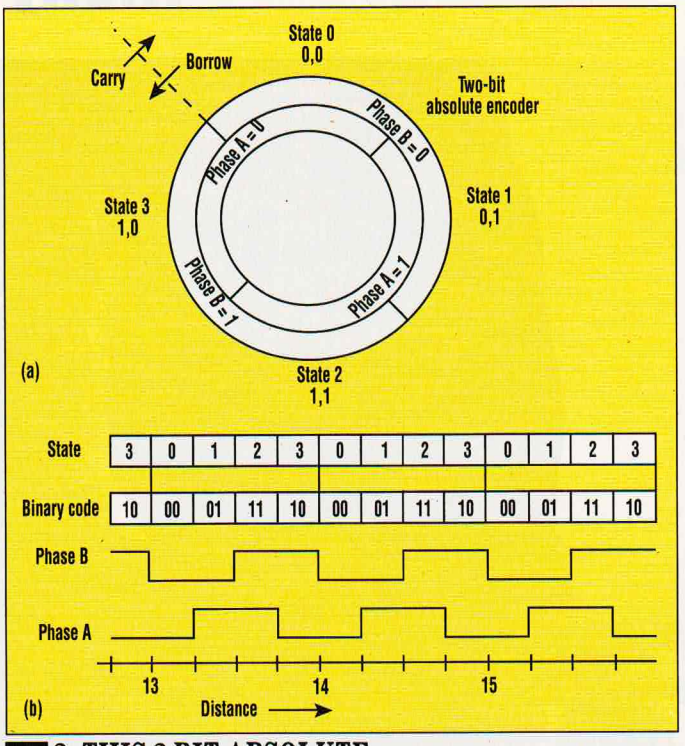
```
                    LISTING 1:
  16-BIT, DOUBLE-RESOLUTION POSITION READOUT
              (INTERRUPT METHOD)

INTSRV  LDA     PORTC       GET BITS
        EOR     #1          INVERT PC0
        STA     PORTC       RESET OUTPUT LINES
*
        LSRA                BRING PC1 TO BIT 0
        EOR     PORTC       COMPARE PC0 WITH PC1
        BIT     #1          TEST RESULT
        BEQ     INTSR2      BR IF PC0=PC1, DECREMENT
*
        INC     POS_LO      LSB
        BNE     INTSR1      BR IF NO ROLLOVER
        INC     POS_HI      MSB
INTSR1  RTI
*
INTSR2  TST     POS_LO      WILL THERE BE A BORROW?
        BNE     INTSR3      BR IF NOT
        DEC     POS_HI      MSB
INTSR3  DEC     POS_LO      LSB
        RTI
```

put doesn't match Phase B's level, the exclusive-OR gate's output is high, and interrupt service isn't being requested. When Phase B changes to match $PC_0$, the gate's output goes low, generating an interrupt request. The service routine resets the interrupt by inverting $PC_0$ to mismatch Phase B. In this manner, Phase B triggers an interrupt on both rising and falling transitions.

An interrupt occurs when the encoder moves between states 0 and 3 or between states 1 and 2 (in either direction). The Phase A input on $PC_1$ identifies which pair of states is being traversed (see the 2-bit absolute encoder). Phase B is determined by the level on $PC_0$. That same level, in conjunction with $PC_1$, determines whether the count is increased or decreased. For normal encoder resolution, the count changes only at Phase B transitions where Phase A is zero. For double resolution, the count changes at every Phase B transition. The program for the implementation in this article uses a double-resolution travel accumulator with 16-bit capacity *(Listing 1)*.

The first three code lines of the program reset the interrupt line. Then, Boolean logic in the next four lines determines whether to increase or decrease the count *(see the table)*. Keep in mind that bit 0 now represents Phase B before it changed. The Count Up line comes from the 2-bit absolute encoder diagram shown in Figure 2. It's easy to see by inspection that Count Up is true when the exclusive-OR of Phase A ($PC_1$) and Phase B ($PC_0$) is true. The program executes in about 25 μs, including interrupt operations, using the usual 4-MHz clock.

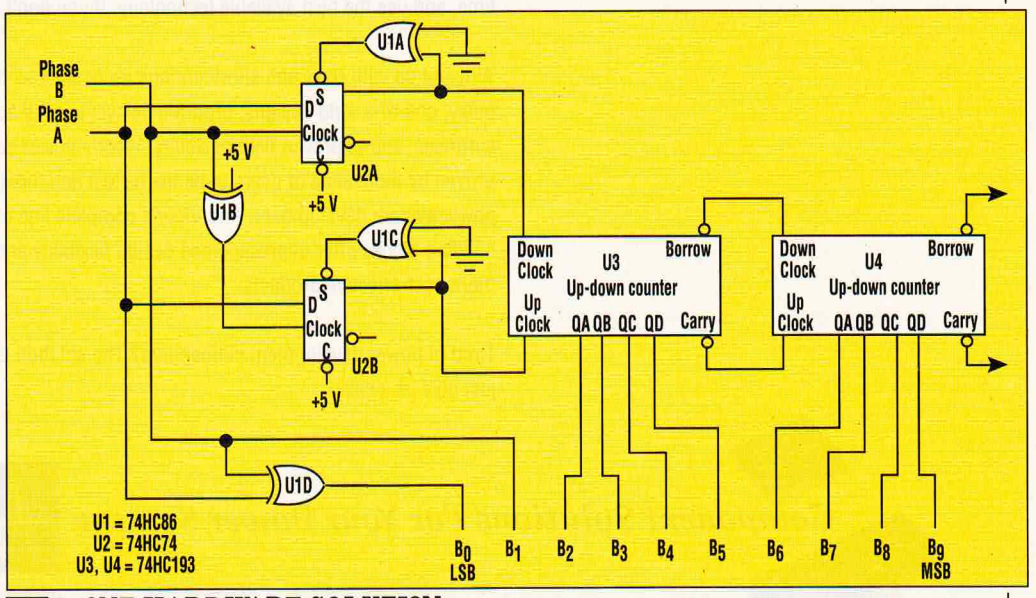



**2. THIS 2-BIT ABSOLUTE** encoder is a shaft encoder that will rotate many times as distance is traversed (a). Phase outputs are interpreted as a binary code with Phase B as the most significant bit (b).

When interrupt operations do not suit the design, polling code may be the best option *(Listing 2)*. In polling, the microprocessor checks the encoder input whenever it has time to spare. As shown in Figure 4, this solution requires no external hardware. It does require, however, that the encoder lines enter the lowest bits of an I/O port for correct positioning in the byte.

The minimum polling interval depends on the mechanism's maximum velocity and acceleration. Although these can be calculated accurately, an empirical check is always wise. To perform the check, the following steps should be taken:

- Run the mechanism with worst-case accelerations (including vibration and shock if relevant).
- Trigger an oscilloscope with one encoder phase, setting its edge at the beginning of the screen.
- View the other phase by itself to avoid the effects of alternating channel sweeps.
- Set the sweep speed to place the average transition in the right half of the screen.
- Observe for 100 or so sweeps to catch the shortest interval between phase edges (a phosphor-type storage oscilloscope is handy for this measurement).
- Then, for thoroughness, repeat the




**3. ONE HARDWARE SOLUTION** for the encoder interface relies on a propagation-delay-reset to produce a narrow clock pulse for the up-down counter.

measurement with the other trigger polarity. The maximum software polling interval equals the shortest trigger-to-edge time observed. Few systems require an interval less than 200 μs.

The polling software constructs a jump vector from the previous and current encoder reading. Because the previous state could have been any one of four states, and the present state could also be any one of four states, there are 16 possibilities to handle.

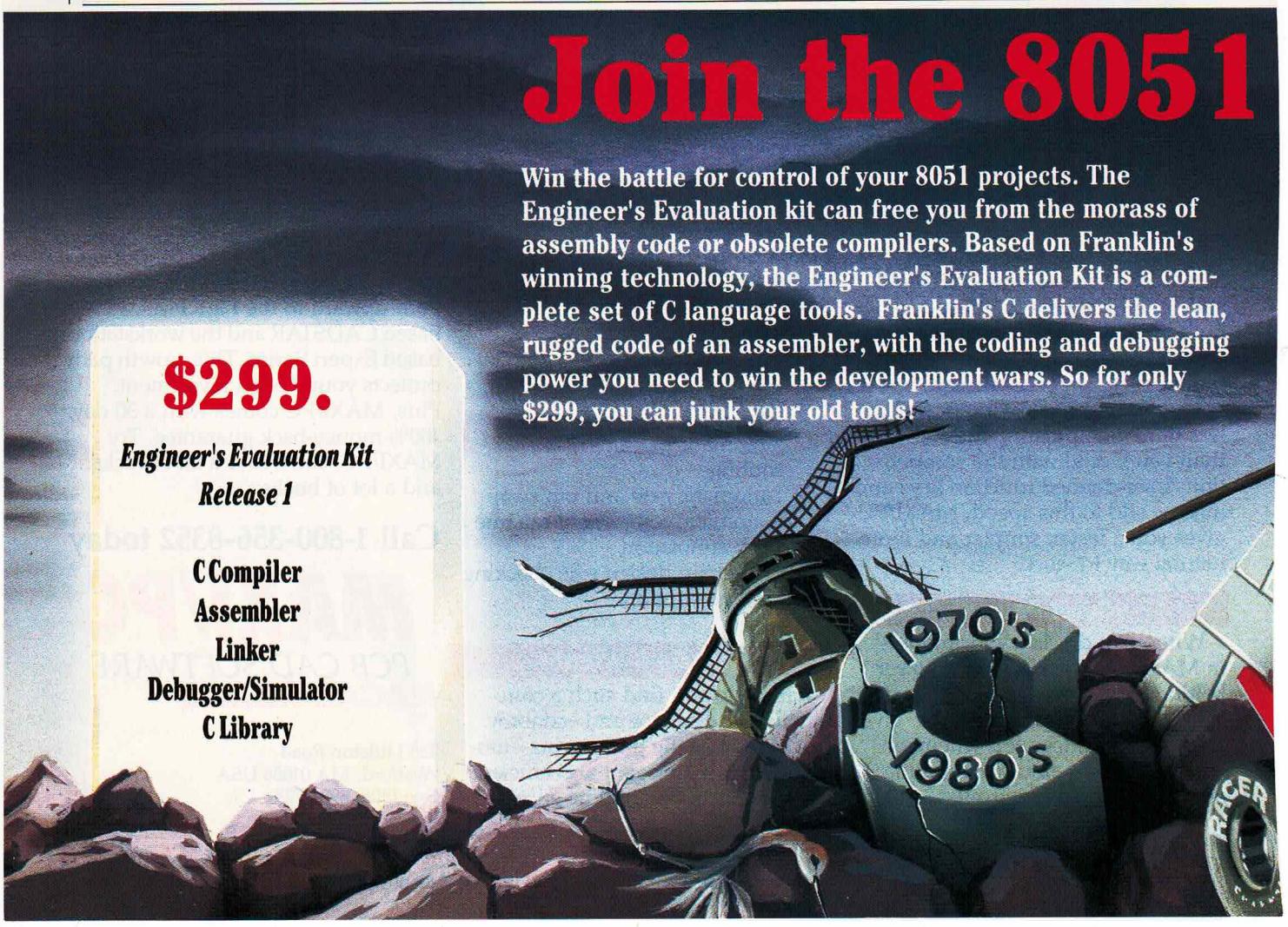
The first steps read the I/O port and zero all but the encoder bits. The previous reading is stored in LAST after shifting zeros into the two lower bits. Adding LAST to the current reading puts the successive readings side-by-side in the byte.

An index table is composed directly from the 2-bit absolute encoder diagram and the 16 binary codes. In four cases, the bit pairs are the same (0000, 0101, 1010, 1111). The state didn't change so the program simply returns from the subroutine. Four more cases indicate non-clocking transitions that also cause the program to return. Two cases (1000 and 0111) must increment and two cases (0010 and 1101) must decrement.

The remaining four cases are impossible

LISTING 2:
16-BIT, DOUBLE-RESOLUTION POSITION READOUT
(POLLING METHOD)

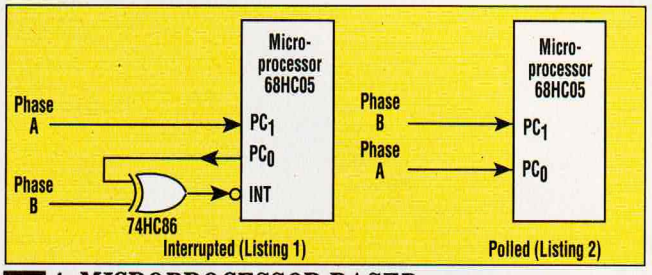
```
ENCPOL  LDA     PORTC       GET CURRENT READING
        AND     #3          MASK TO ENCODER BITS
        ADD     LAST        ADD 4X PREVIOUS STATE
        AND     #15         FORM 4-BIT INDEX
        TAX                 A→ X FOR INDEXING
        LSRA                MUL BY 4
        LSRA                CLEARS 2 LOW BITS
        STA     LAST        SAVE FOR NEXT TIME
        LDX     ETBL,X      GET OFFSET
        JMP     ENC,X       CONTINUE THERE
*
ETBL    FCB     ERET-ENC,ERET-ENC,EDEC-ENC,EERR-ENC
        FCB     ERET-ENC,ERET-ENC,EERR-ENC,EINC-ENC
        FCB     EINC-ENC,EERR-ENC,ERET-ENC,ERET-ENC
        FCB     EERR-ENC,EDEC-ENC,ERET-ENC,ERET-ENC
*
ENC     EQU     *           EQUATE ENC WITH CURRENT ADDRESS
EINC    INC     POS_LO      INCREMENT POSITION COUNT
        BNE     EINC1       BR IF NO CARRY
        INC     POS_HI
EINC1   RTS
*
EDEC    TST     POS_LO      DECREMENT POSITION COUNT
        BNE     EDEC1       BR IF NO BORROW
        DEC     POS_HI
EDEC1   DEC     POS_LO
        RTS
*
EERR    BSET    0,ERROR     STATE MISSED! SET FLAG
ERET    RTS
```



transitions because the states aren't adjacent. If one of these occurs, the polling missed a state change. If the undetected traverse was by the clockwise path, 1 must be added to the count. If counterclockwise, 1 must be subtracted. However, because there's no way to tell which path was taken, the count is off by one and an error must be flagged. The polling program in this article finishes in 21 to 29 μs with a 4-MHz microprocessor clock.

There are many helpful tips to keep in mind when working with encoders. Numerous practical encoder applications show Schmitt-trigger buffers between the encoder and logic. Are these necessary? For the circuits above, they aren't needed if the input slew rate exceeds 2 V/μs. That's because the new CMOS (HC or HCT) gates typically have enough gain to accept slew rates as low as 0.04 V/μs without error. The exclusive-OR gates, however, handle low slew rates poorly because the internal logic uses a low-gain input gate.

On a final note, two-fold resolution increases are fairly trouble free, but four-fold increases may not be practical for closed-loop servo control. Although the encoder's specifications imply four distinct states, they aren't guaranteed to be of equal widths. The servo may not be able to stop within a narrow state that meets minimum tolerances. This would cause continuous oscillation between the adjacent states. For simple position readouts, however, the "missing" state isn't usually a problem.□




**4. MICROPROCESSOR-BASED** encoder interfaces are driven by either interrupt or polling software schemes. The interrupt driver requires additional external hardware.

*James A. Kuzdrall is a designer of instrumentation and automated process controls at Intrel Service Company. He holds an SB in physics from the Massachusetts Institute of Technology, Cambridge, and a Master of Science in engineering from Northeastern University, Boston, Mass.*

| HOW VALUABLE? | CIRCLE |
|---|---|
| HIGHLY | 535 |
| MODERATELY | 536 |
| SLIGHTLY | 537 |

C Revolution!

Nothing beats the right tools-- in power, speed, code efficiency, or ease of use, the Professional Developer's Kit commands success. It includes the industry's top rated 8051 C Compiler, high-level language debugger/simulator, banking linker, and the first multi-tasking RTOS small enough for single chip solutions. The advanced GUI and mouse control both the simulator and debugger, as well as your Intel or Nohau 8051 emulator. Franklin's best dominates where other tools fail!

*For Quick Action, call the 24 Hour Hotline:*
*(408) 296-8056*
*Please have your FAX number ready.*

$2495.

*Professional Developer's Kit Release V*

Optimizing C Compiler
Real Time Multitasking OS
Graphical User Interface
HLL Degugger/Simulator
Banked Linker
Macro Assembler
Extended C Library

FRANKLIN SOFTWARE, INC
888 SARATOGA AVE., #2, SAN JOSE, CA 95129
PHONE: (800) 283-4080 EXT. 890

CIRCLE 218 FOR U.S. RESPONSE
CIRCLE 219 FOR RESPONSE OUTSIDE THE U.S.