HOW TO FULLY DEBOUNCE LOW-COST KEYBOARDS

Inexpensive approach to properly interfacing calculator keyboards to microcomputers

BY RALPH TENNY

KEYBOARDS for home-brew calculators or even minicomputers are available from many sources (such as those advertising in this magazine). Unfortunately, most of these keyboards lack two important features—contact debouncing and key encoding. Without debouncing, each key closure can produce multiple signals, while encoding makes it possible to determine just which key has been operated.

There are many different types of debouncing circuits, but most of them are applicable to just one contact. Debouncing a full keyboard can be very expensive. Also, when debouncing a keyboard, encoding is very complex since each key closure must result in a unique code output.

A simple, low-cost way to overcome these problems is shown in the circuit in Fig. 1. It produces a binary-coded output, fully debounced, from any low-cost 16-key board. The keys are labelled in hexadecimal so that the board can communicate with a microprocessor.

Four 8-input NAND gates (*IC1* through *IC4*) encode all the keys except 0. All of these gates are held high by resistors *R4* through *R19*. When a key is depressed (except 0), that input is brought to ground and its associated NAND gate output goes high. For example, if key D is depressed, the outputs of *IC1*, *IC3*, and *IC4* go high to produce 1101, the hex code for D.

Because it is necessary to know when any key, including 0 is depressed, OR gates *IC6A* and *IC6B* detect the presence of any key closure. When a 0 comes on, *IC5A* passes the signal to gate *IC6B*.

Although we can now detect all key closures and encode them on the 1-2-4-8 lines, contact bounce remains a problem. The waveforms in Fig. 2 show what happens when a key is depressed and bounces one time. The output of *IC6B* is waveform M, which drives the *R1-C1* combination to produce waveform N. When the key stops bouncing, *C1* can be charged up enough to cause *Q1* to fire. Resistor *R1* is low enough so that *Q1* latches in and stays on. As a result, point P is held low, and the two sections of *IC5* produce high or low Data Ready signals. The latter is indicated by a rule over the words in Fig. 1.

When *Q1* latches in, it provides some protection against multiple-key closures. If a second key is depressed after *Q1* fires, the output code will be correct but no Data Ready strobe will be produced.

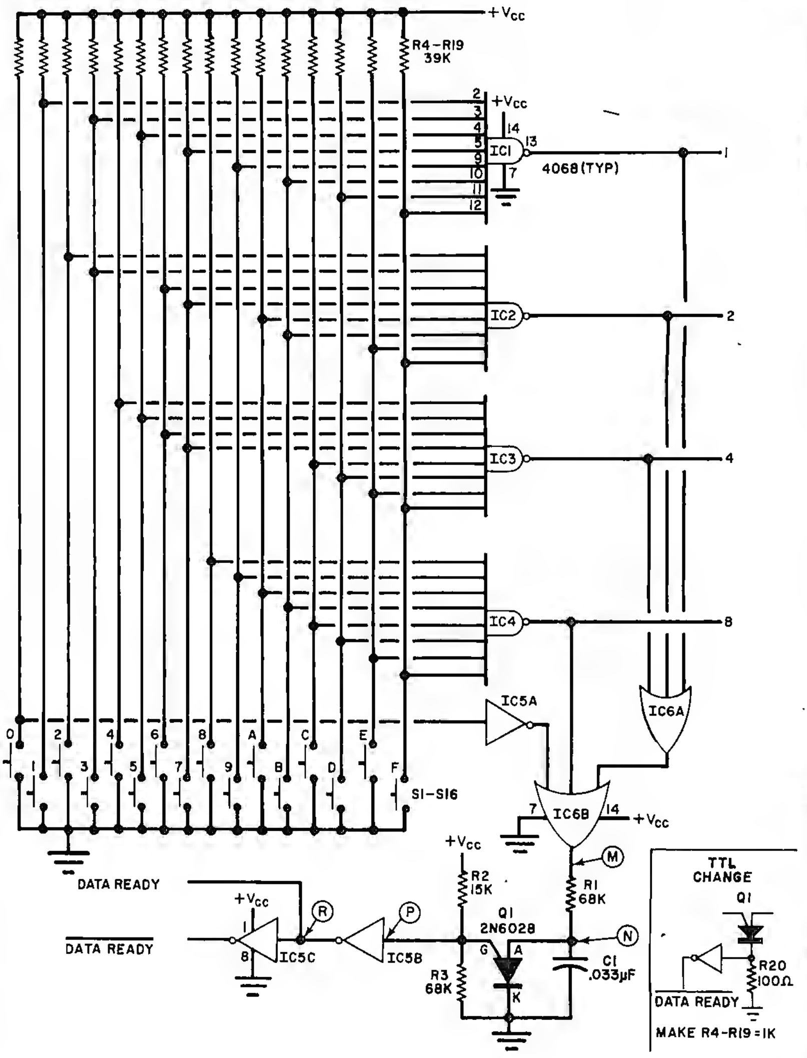

Fig. 1. The circuit produces a 1-2-4-8 binary code and provides a debounced data-ready strobe.

PARTS LIST

C1—0.033-µF, 15-V capacitor
IC1-IC4—CD4068 8-input NAND gate (for TTL, use 7430)
IC5—CD4049 inverting hex buffer (for TTL, use 7404)
IC6—CD4075 triple 3-input OR gate (for TTL, use 7432)
Q1—2N6028 programmable unijunction transistor
R1,R3—68,000-ohm, ¼-watt resistor
R2—15,000-ohm, ¼-watt resistor
R4-R19—39,000-ohm, ¼-watt resistor
R20—100-ohm, ¼-watt resistor (TTL only)
S1-S16—Normally open switch/keyboard

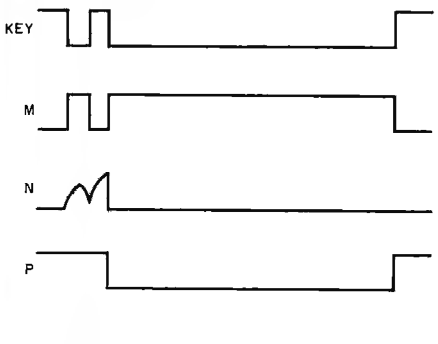

Fig. 2. Waveforms show how bouncing key can produce a clean single output.

The UJT will reset only after all the keys are released, and the output of *IC6B* returns to zero. If a second key is closed within a few milliseconds (while the first key is still bouncing), an erroneous output can be produced.

The keyboard can be battery powered if the CMOS devices are used. If you are going to drive TTL logic with this adaptor, change the IC's to their TTL counterparts (see Parts List), change the values of *R4* through *R19* to 1000 ohms, and add a 100-ohm resistor in the cathode of *Q1*. The Data Ready signal is then available as shown in Fig. 1. ◇