William L. Switzer
Department of Chemistry
North Carolina State University
Raleigh, N.C. 27607

Asynchronous Serial Computer Interfaces

In their REPORT in ANALYTICAL CHEMISTRY, Dessy and Titus (*1*) reviewed many problems associated with computer interfacing. They discussed trade-offs in selecting and/or designing an appropriate interface. Particularly noteworthy was their "TTL" principle which describes the trade-offs among *T*ime, *T*alent, and *L*ucre. The reason often given for designing an interface oneself is to save money. But building an interface from components is inexpensive if and only if the costs associated with time and talent can be neglected or at least amortized over a number of projects. A better reason may be that the experience gained in designing one's own interface can greatly simplify future interfacing problems.

With even minimal knowledge of the philosophies of digital data transmission, the scientist with some background in digital electronics can design in block diagram fashion a digital data link between his laboratory instrument and a computer or a teletypewriter. On considering his block diagram in more detail, he will likely become discouraged at the straightforward but extensive circuitry required. Designing such a circuit from small- and medium-scale integrated circuits is not practical for most novice circuit designers. Starting from large-scale integrated circuits or multicomponent circuits will still offer many design challenges. Herein lies the purpose of this article, to examine the problems and some solutions for designing an interface between the user's laboratory instrument and a computer or teletypewriter. This article deals with a very specific, but generally applicable, type of interface in which the user's parallel TTL (*T*ransistor *T*ransistor *L*ogic) output is transmitted in an asynchronous serial fashion.

Data Codes

Before proceeding, let us review the concepts of a digital code and the two methods for transmitting that code: parallel and serial. (This subject is discussed extensively in ref. *1*). A digital code is a string of *bi*nary digit*s*, or *bits*. This string of bits, logic 0's and 1's, has a unique meaning to the interpreter. The three most popular codes are binary, *B*inary *C*oded *D*ecimal (BCD), and *A*merican *S*tandard *C*ode for *I*nformation *I*nterchange (ASCII).

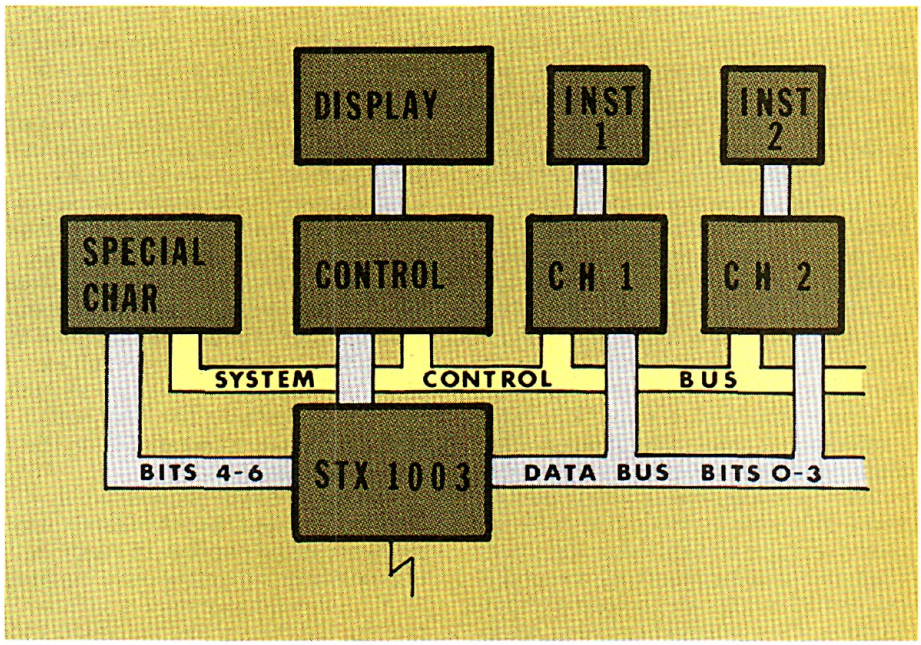

Figure 1. Multichannel asynchronous serial transmitter
Block diagram shows two instruments connected to two active channels

In Table I the decimal number 15 is shown in each code. In each case there is a one-to-one relationship between the decimal number and its digital code. The binary code is simply the base 2 representation of the number which is 1111_2; this code requires the fewest bits for representing any number. The BCD representation for 15 is the base 2 representation of each decimal digit and is $0001_2\ 0101_2$; this code is the easiest for converting between decimal and BCD. The ASCII representation precedes each BCD digit with the three bits 011_2 (for numeric data); thus, 15 is $0110001_2\ 0110101_2$. The extended word length of the ASCII code allows one to represent as a string of bits, not only numeric characters, but also alphabetic and special characters.

These digital codes should not be confused with octyl (base 8) and hexadecimal (base 16) codes which are merely shorthand representations of the binary codes shown in Table I. Digital interfaces always involve only two "levels" of information, logic 0 and logic 1, *not* 8 or 16 as might be inferred from the use of these bases.

Parallel vs. Serial Transmission

If each bit of the digital code is simultaneously available, the code is presented in parallel. If each bit is sequentially available (one bit at a time) starting with the least significant bit, the code is presented serially. With serial data exchange, timing becomes especially important since each bit in the code must be transmitted and received during a finite time interval from the beginning of the transmission. Data exchange between transmitter and receiver is synchronized using digital timing techniques. Two

Table I. Digital Codes for Decimal Number 15

Code	Binary (base 2) representation	Octyl (base 8) representation	Hexadecimal (base 16) representation
Binary	1111_2	17_8	F_{16}
BCD	0001_20101_2	025_8	15_{16}
ASCII	$0110001_20110101_2$	061_8065_8	$31_{16}35_{16}$

methods are commonly employed: synchronous and asynchronous serial data exchange. Synchronous receivers and transmitters use a common clock, while asynchronous devices use separate clocks operated at the same frequency. Synchronous devices transmit each bit only during specified pulses of the clock, while asynchronous devices transmit data whenever it becomes available. For this reason asynchronous transmission is more suitable for interfacing both chemical instrumentation and devices such as keyboards, which produce data randomly with respect to any clock pulses. A more detailed comparison of the two methods of serial data exchange is discussed by Finkel (2), but further consideration here will be limited to asynchronous serial data exchange.

Consider the problem of transmitting 8 bits of digital data. In parallel transmission, 8 transmission lines plus a reference line (9 total) are required; in serial transmission, only 1 line plus a reference (2 total) are required. The main advantage of serial transmission is that it requires only a single pair of wires. When transmitting across a room, wiring 9 transmission lines and their associated connectors is practical, but when transmitting to a remote location, dual-line transmission is usually more practical. Even when transmitting short distances, if the data are transmitted to different receivers on different occasions, the scientist will quickly tire of wiring multipin connectors. Thus, considering the number of transmission lines, serial transmission is simpler.

Disregarding for the moment the obvious requirement of more complex electronics to "catch" data "on-the-fly", another obvious disadvantage of serial transmission is that data are transmitted more slowly. At first glance it should take eight times longer to transmit 8 bits serially than in parallel. The situation is actually somewhat worse. Consider the serial transmission mode used by a teletypewriter. In addition to a 7-bit ASCII character, a parity bit, a start bit (signaling the beginning of transmission), and two stop bits (a "rest" interval between transmissions) are required. Thus, a total of 11 bits is transmitted in order to transmit the 7-bit code. One loses about a factor of 10 in the transmission speed.

Returning to the illustration in Table I, one sees that the fastest way to transmit the number 15 requires transmitting the binary code in parallel. The slowest way requires transmitting the corresponding ASCII code in a serial fashion; this method requires 22 times longer (2 characters × 11 bits each), presuming the serial and parallel receivers accept bits at the same rate. Nevertheless, transmitting serial ASCII code has the advantage of being nearly universally accepted by teletypewriters (if transmitted at 110 baud) as well as nearly all computers. (The baud rate is the transmission rate; 110 baud means 110 bits/s, *but* these bits include the 7-bit code, the parity bit, the start bit, and two stop bits. Each character requires that 11 bits be transmitted; thus, one character is transmitted every 100 ms at 110 baud.) With many computers the baud rate may be greatly increased, thus increasing the transmission speed. Besides being compatible with computer hardware, serial ASCII is also compatible with most computer software. Many Basic and Fortran compilers can perform I/O directly in the ASCII code. Considering both hardware and software interfacing problems, serial ASCII data transmission is highly desirable despite its slowness and possible problems in generating the ASCII code.

Thus far, primarily data transmission has been discussed. The reverse operation, data reception, must also be considered. Herein the serially

transmitted string of bits must be received "on-the-fly" and converted to a parallel format. Any start and stop bits, which were added on transmitting, must be removed on receiving; also, the parity must be checked, and an appropriate error output set.

Universal Asynchronous Receiver/ Transmitter—UAR/T

Although simple in principle, transmitting and receiving data serially require rather sophisticated electronics. The novice would be wise to avoid starting from basic TTL gates. However, a large-scale integrated circuit called a UAR/T (General Instrument Corp., 600 W. John St., Hicksville, N.Y. 11802) is available which sacrifices little in general applicability. In fact, it would be both time consuming and expensive for the novice to design such a versatile system from scratch.

The UAR/T consists of separate transmitter and receiver sections. The transmitter accepts bit-parallel input and produces bit-serial output; the receiver accepts bit-serial input and produces bit-parallel output. The transmitter and receiver may be operated independently of each other (full duplex) or in series (half duplex), meaning that whatever is transmitted is also received. The number of bits in the code can be selected (5–8 bits; ASCII is a 7-bit code). An optional parity bit may be added to the code when transmitting or deleted when receiving (an error flag is set if appropriate), and the parity may be selected even or odd. The UAR/T also adds the start bit and stop bit(s) (one or two stop bits may be selected) when transmitting and deletes them when receiving. The UAR/T also provides two status outputs, one indicating that the transmitter buffer is empty, the other indicating that the receiver buffer is full. The baud rate is determined by an external clock.

All inputs and outputs on the UAR/T are TTL compatible. Most modern devices, which are likely to be interfaced with the parallel inputs and outputs, will be TTL compatible. However, serial input and output devices are often not TTL compatible. The two most popular logic conventions are current loop and RS-232. Current loop, which is used by conventional teletypewriters and, hence computers with teletype interfaces, specifies that logic levels 0 and 1 are nominally 20 and 0 mA, respectively. RS-232, which is used by most modems, specifies that logic levels 0 and 1 are +5 to +15 and −5 to −15, respectively. (Modem is a device which codes and decodes logic 0 and 1 as two different frequencies. This is generally used for long distance data links.)

Converting between TTL logic levels and current loop or RS-232 and vice versa is quite easy. Larsen and Rony (*3*) give simple circuits for accomplishing this conversion. With TTL, current loop, or RS-232, interfacing can nearly always be accomplished.

For transmitting a single character serially, the UAR/T requires little more than connecting the parallel inputs and starting transmission, while receiving a single character requires little more than sampling the parallel outputs when the device indicates data are available. However, most scientists wish to transmit more than a single character. The UAR/T requires that each character (or word) be loaded in a word-serial, bit-parallel fashion. Building a transmitter/receiver from a UAR/T still requires considerable electronic design to load in this fashion. For many applications, such as transmitting data from a digital panel meter, the data are available in a word-parallel, bit-parallel fashion. The designer must therefore convert from this word-parallel format to a word-serial format. On command to

(*Continued on page 1010 A*)

load his transmitter, the data must be saved, and the first character must be transferred to the parallel inputs on the UAR/T and transmitted. Then each successive character must be routed to the parallel inputs and transmitted when the previous transmission is complete. Although simple in principle, designing such a circuit can be quite time consuming.

*SER*ial *D*ata *EX*change—SERDEX

A device which more closely meets the usual laboratory requirement of transmitting word-parallel, bit-parallel data is the STX 1003 SERDEX (Analog Devices, P.O. Box 280, Norwood, Mass. 02062). This device, which was described by Larsen and Rony (*4*), retains many of the features of the UAR/T. It has separate transmitter and receiver sections and can be operated either full duplex or half duplex. The number of bits per word can be selected from 5 to 8; parity is optional and can be selected either even or odd. The start and either one or two stop bits are added to the code. The baud rate is determined by an external clock. The STX 1003 also provides switching circuitry for interfacing by current loop.

A companion component is the SCL 1006 clock module which provides the appropriate clock pulses for selecting baud rates from 110 Hz to 19.2 kHz. From a +5-V power supply, it also produces −15 V, which is required by the STX 1003. Thus, the SCL 1006 and STX 1003 may be operated from a single +5-V power supply.

Considering first the STX 1003 as a transmitter, one sees that the principal difference between the UAR/T and the SERDEX is the ability to load in full parallel, i.e., word-parallel, bit-parallel, up to 8 words each having 4 bits. This ability is sufficient to load up to 8 BCD or octyl characters in full parallel as provided by most digital output devices. Following a load command, the data are latched and then transmitted 4 bits of latched data per character. Transmission continues until a 1111_2 is detected; this character serves as a STOP character. Transmitting until a special character is detected simplifies the use of this device in some respects, but complicates its use in others.

The advantages and disadvantages arising from the STOP character will be discussed further below. But returning to the character makeup, one sees that each of the 4-bit words which were loaded in parallel is preceded by 1–4 bits depending on the number of bits of code selected and on whether parity is generated. (Preceded here means the bits are more significant; they are, in fact, transmitted last since the least significant bits are transmitted first.) The added bits may be loaded in a word-serial (not parallel), bit-parallel fashion. For most interfacing applications, which require transmitting numeric data only, loading the most significant bits in this fashion causes little difficulty. Consider the problem of transmitting an ASCII code for a BCD digit. As noted earlier, the ASCII code for numeric data is the BCD code preceded by 011_2. The three most significant bits are always the same and, hence, may be "hard-wired" to the appropriate logic level.

When attempting to transmit any data other than numeric data, problems arise requiring user-designed circuitry. Then one must somehow load the appropriate 3-bit prefix in a word-serial, bit-parallel fashion. One may either add three shift registers which allows him to load the entire 7-bit code in a word-parallel, bit-parallel fashion or attempt to generate the appropriate 3-bit prefix as each word is transmitted. Appropriate synchronization signals are provided to facilitate either approach. Adding shift registers is the most versatile approach for expanding the character set; however, this requires adding 24 (3 × 8) parallel input lines to the 32 (4 × 8) already required. Generating the 3-bit prefix "on-the-fly" is usually preferred if only a few nonnumeric characters are required.

The nonnumeric characters, which are frequently useful, are + and − signs, *C*arriage *R*eturn (CR) and *L*ine *F*eed (LF), comma, and space. The "SERDEX User's Guide" (*5*) is helpful in describing how to transmit a sign; the ASCII characters for + and − require a prefix of 001_2. This reference also describes a convenient manner for producing CR/LF which requires a prefix of 000_2. Lorimer and Bell (*6*) describe a system in which they sample and transmit a digital panel meter nine times before producing a CR/LF. With minor modifications this circuit could transmit any number of samples per line. In their circuit, each transmission is separated from the previous one by a comma, which requires a prefix of 010_2. With minor modification the comma could be changed to an ASCII space character, which requires the same prefix. Adding one or two nonnumeric characters to a numeric character set is relatively easy as long as they occur at exactly the same point in each transmission (preferably first or last).

Returning now to the use of a special character to stop transmission, one can see both advantages and disadvantages. As mentioned earlier, a 1111_2 serves as the STOP character. Depending on which of several 3-bit prefixes precede this, several characters in the ASCII set will terminate transmission. On the one hand, having even one character terminate transmission is a disadvantage since transmitting the entire ASCII character set is difficult, although not impossible, since one can inhibit detection of the STOP character. On the other hand, having more than one character stop transmission is an advantage, since the user can stop transmission with one of several characters which can be generated under different conditions. Depending on which STOP character is detected, the user's computer can tell which condition caused termination.

On examining the SERDEX more closely, the STOP character is really $X1X1111_2$ where X may be either 0 or 1. The STX 1003 changes this to a $X0X1111_2$, if detection of STOP has not been inhibited. The reason can be readily seen. Since the three most significant bits will usually be 011_2 for numeric data, the resulting ASCII character is nonprinting, if the STX 1003 changes these to 001_2. This means that with a teletypewriter the STOP character will be deleted from the printed output.

However, other problems with the STOP character can still arise, as Lorimer and Bell (*6*) point out. In transmitting to their computer, which is programmed in Fortran, the STOP character is illegal and causes an input error. These authors describe a method for deleting the STOP character by shorting the current loop outputs from the STX 1003 when STOP is detected.

Considering now the STX 1003 as a serial receiver, one finds very little difference between the SERDEX and the UAR/T. Both provide word-serial, bit-parallel output. However, the STX 1003 also decodes certain special characters providing an open-collector, pulsed output when anyone of these is detected. These special characters include: ?, *, %, !, ′, =, and $. These characters generate useful output pulses for controlling the user's instrument.

Multichannel Data Transmission

As already discussed, the UAR/T requires little additional circuitry to transmit one ASCII character. Similarly, the STX 1003 requires little additional circuitry to transmit as many as eight ASCII characters as long as only numeric characters are allowed. But for transmitting more than one channel having as many as eight characters each, the user must design his own circuit. In our laboratory, we built such a multichannel transmitter. With this device, we wanted to transmit channel-parallel, word-parallel, bit-parallel data. Desiring to minimize the design effort, we chose the STX 1003 as the basis for this interface. Using

Table II. Character Set

LF	0	0	0	0	1	0	1	0
CR	1	0	0	0	1	1	0	1
+	0	0	1	0	1	0	1	1
−	0	0	1	0	1	1	0	1
EOT	1	0	1	0	1	1	1	1
STOP	0	0	1	1	1	1	1	1
NUMERIC	X	0	1	1	X	X	X	X

the STX 1003 requires scanning each active channel to load and transmit data. The system is constructed so that the user's laboratory instrument communicates with a data acquisition computer, an IBM SYS/7 in our case. One may substitute a teletypewriter for the SYS/7, allowing the same instrument control from a keyboard. For obtaining a printout of data and for debugging computer-controlled data acquisition, this teletypewriter compatibility has proved valuable.

Figure 1, which shows a block diagram of the system, illustrates the complexity of a multichannel transmitter. The master control coordinates the interactions of all other components in the system. The special character generator and each channel are connected via a common control bus. The STX 1003 and display are connected via other lines. On receiving an inquiry, which is the ASCII character for "?", the STX 1003 produces a pulse which starts the sampling sequence. From each active channel, the master control waits for a return signal indicating valid data at the inputs.

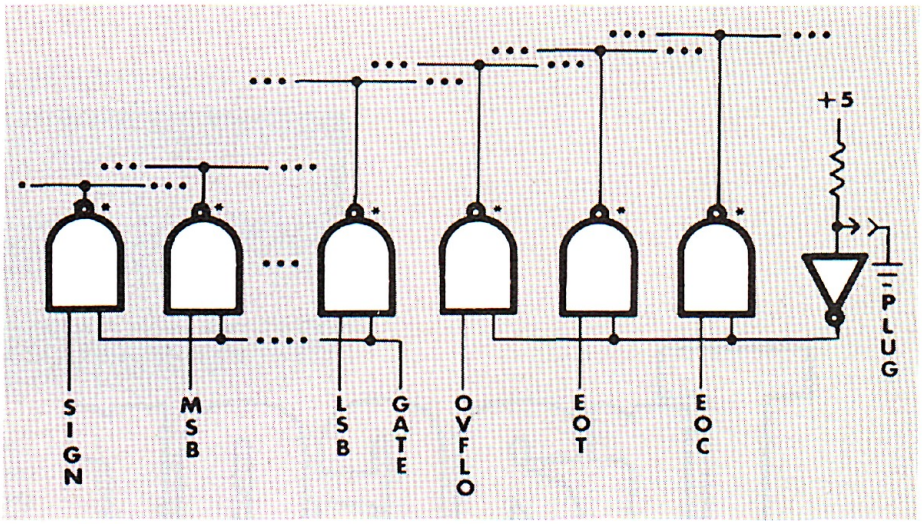

Figure 3. Data bus routes input data into STX 1003 parallel inputs and control indicators into appropriate control logic

One set of open-collector gates required for each channel. SIGN: HI for +; MSB → LSB: parallel input data; GATE: HI to route data onto bus; OVFLO: HI to signal overflow condition; EOT: HI to signal end of data acquisition sequence; EOC: HI to signal conversion complete; PLUG: activates channel when grounded by input connector

Only those channels connected are queried and transmitted. Each channel is routed sequentially onto a data bus. The system has been built with only two channels, but is expandable, and the control logic has been breadboarded with four channels. The control block also activates appropriate indicator lights on the display panel and accepts certain manual override inputs.

Table II shows the allowed character set, which consists of LF, CR, +, −, *E*nd *O*f *T*ransmission (EOT), STOP, and numeric data. The EOT character differs from the STOP in that it signals the SYS/7 to terminate a run. Note that we have taken advantage of having multiple STOP characters. Both EOT and STOP terminate transmission from the STX 1003, but STOP only indicates to the SYS/7 that transmission of each channel is complete, while EOT signals the end of a data acquisition sequence. Except for the +, −, and CR, the lowest order four bits are loaded from the data bus. The + and − signs are generated using circuitry suggested in the "SERDEX User's Guide" (5). The lowest order four bits for a CR are identical with those for the − sign and thus produced as if for a − sign. The special character generator, shown in Figure 2, produces the three most significant bits of the ASCII code excluding parity. By default, this circuit produces 011_2 for numeric data and STOP. But for CR and LF it produces 000_2 and for +, −, and EOT, 010_2. Parity is generated internally by the STX 1003. Figure 2 serves to illustrate how complicated the circuitry becomes to produce even this limited set of nonnumeric ASCII characters.

Figure 3 shows the data bus over which data and control logic from each channel are routed to the STX 1003. A channel is activated by grounding a single pin on the input connector. Overflow, end of transmission, and end of conversion are held HI if the channel is not active. These are the no overflow, no end of transmission, and conversion complete conditions. If a channel is activated, each control input must be held at the appropriate logic level by the input device. The sign and each bit are sequentially gated onto

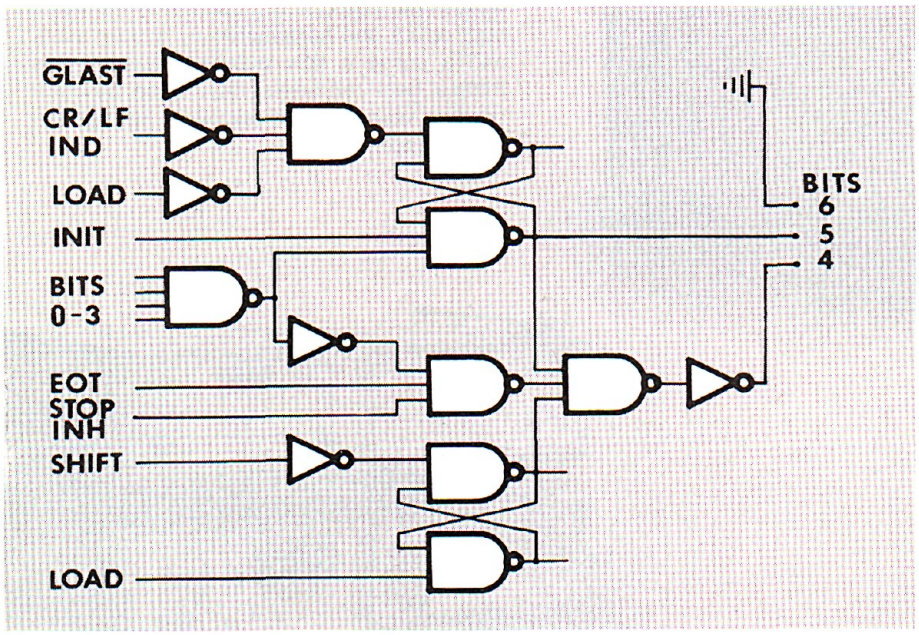

Figure 2. Special character generator

Generates bits 4, 5, and 6 of ASCII code. GLAST: LO when last active channel loaded; CR/LF IND: LO when CR/LF plug connected; LOAD (twice): LO to latch STX 1003 inputs for transmission; INIT: LO to initialize all counters and flip flops; bits 0 → 3: four least significant bits of ASCII code; EOT: HI to signal end of data acquisition sequence; STOP INH: LO to inhibit detection of STOP; SHIFT: positive edge advances STX 1003 to next latched character; bits 4 → 6: three most significant bits of ASCII code

Table III. Channel Gate Truth Tables

	G1	G2	G3
ONE CHANNEL	1	0	0
TWO CHANNELS	1	0	0
	0	1	0
	1	0	0
THREE CHANNELS	1	0	0
	0	1	0
	0	0	1
	1	0	0

the data bus by the channel gate.

Table III shows the truth tables for the channel gates. These truth tables depend on the number of activated channels. If only one channel is connected, then channel one is always active and all other channels are inactive. If two or more channels were connected, then each would be gated onto the data bus and following transmission of the last channel, the gate configuration would return to its initial state. This gating sequence is accomplished through an expandable ring counter and a retriggerable RS flip flop, the channel scanner.

Figure 4 shows the expandable ring counter. The first channel requires one JK flip flop and two NAND gates; each other channel requires one JK flip flop and one open-collector NAND gate. Outputs G1 and G2 are the gate signals for channels 1 and 2, respectively. If only one channel is connected, G1 remains HI and G2 remains LO on the positive edge of a clock pulse produced by the *chan*nel *scan*ner (CHAN SCAN). If, however, a second channel is connected, G1 begins HI while G2 is LO, but on the first positive edge of CHAN SCAN, G2 goes HI and G1 goes LO. On the second positive edge G1 returns HI and G2 returns LO.

The channel scanner, which generates the clock pulse to advance the channel gates, must produce one pulse if only one channel is active, two if two are active, etc. Figure 5 shows the circuit which accomplishes this. It consists of two monostable multivibrators and one RS flip flop made from NAND gates. The RS flip flop, which is the channel scanner output, is initially set to HI. On an end of conversion (all conversion complete), monostable M1 clears the RS flip flop. This loads the STX 1003 and starts the transmission. The STX 1003 sends a shift pulse indicating transmission has begun; this resets the RS flip flop. If monostable M2 is activated, it again clears the RS flip flop on the positive edge of $\overline{\text{XMIT}}'$. (XMIT is an output from the STX 1003 which is HI during

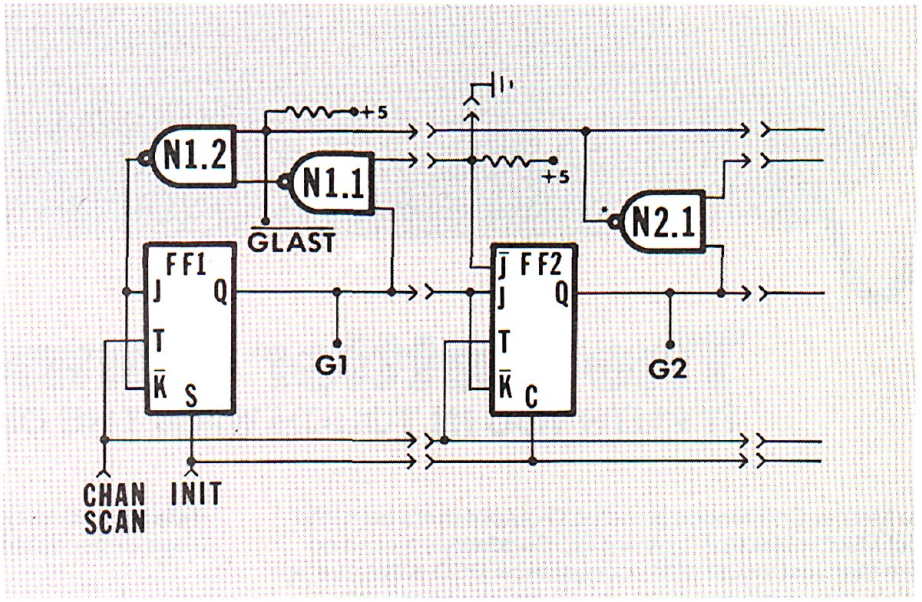

Figure 4. Expandable ring counter

Generates gating pulses for sequentially routing active channels onto data bus. CHAN SCAN: clock pulse which advances channel gate; INIT: LO to initialize all counters and flip flops; G1 and G2: HI to route data onto bus; $\overline{\text{GLAST}}$: LO when last active channel gated onto bus

transmission. XMIT′ is generated with external circuitry and remains HI long enough for transmitting one additional ASCII character. ($\overline{\text{XMIT}}'$ is the complement of XMIT′.) If only one channel is connected, G1, the gate pulse to channel 1, remains HI. This inhibits M2 from triggering on $\overline{\text{XMIT}}'$ and leaves CHAN SCAN HI. If, however, more than one channel is connected, G1 goes LO on the first positive edge activating M2 for a positive edge transition of $\overline{\text{XMIT}}'$. Monostable M2 clears the RS flip flop which loads the second channel and starts its transmission. Alternately, the channel gate will be advanced and transmission will begin until G1 again returns HI, indicating that all channels have been transmitted. M2 will again be inhibited until a new end of conversion is detected.

This data acquisition system has served our laboratory well. It ap-

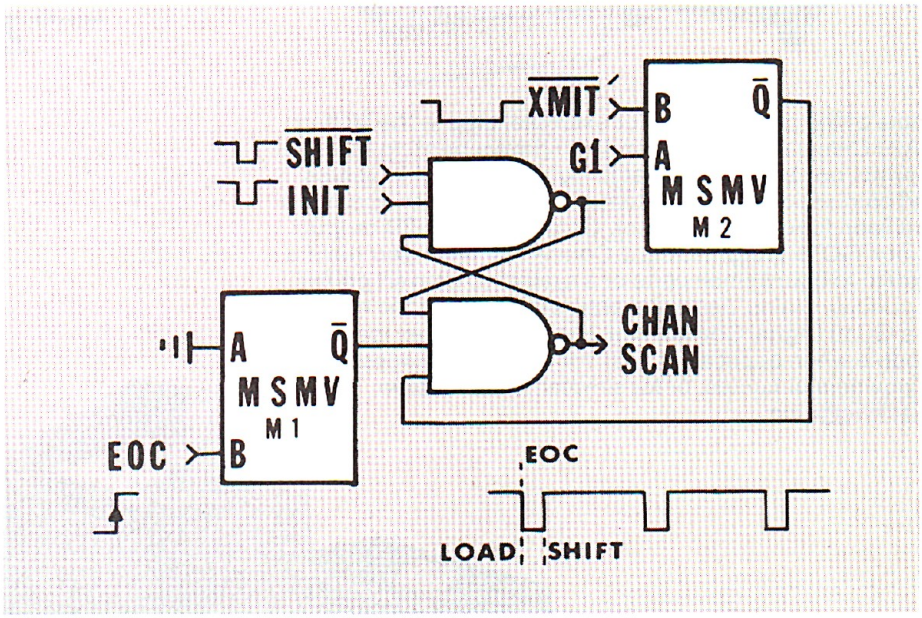

Figure 5. Channel scanner

Serves both to trigger STX 1003 for beginning transmission of each active channel and to advance channel gate for routing active channels onto data bus. EOC: positive edge signals all conversions complete; $\overline{\text{SHIFT}}$: negative edge advances STX 1003 to next latched character; INIT: LO to initialize all counters and flip flops; $\overline{\text{XMIT}}'$: positive edge signals STX 1003 available for reloading; G1: HI when first channel gated onto data bus; CHAN SCAN: negative edge loads STX 1003, positive edge advances channel gate

proaches very closely the "ideal" computer interface with which the users can wire for each channel a single, parallel connector between his instrument(s) and this system and operate his instrument under control of the device linked serially with the STX 1003. Besides operating this interface with teletypewriter and IBM SYS/7, we have also operated it with an Altair 8800 microcomputer. We have used the interface both full duplex and half duplex. We also have a limited capability for selecting the baud rate through control logic.

Future Serial Computer Interfaces

But despite the versatility of the system built around the STX 1003, microprocessors will undoubtedly play an important role in future serial computer interfaces. With an 8-bit microprocessor and its clock, a parallel I/O port, and a serial I/O port, one could eliminate the need for almost any electronic design. Under program control, one could select and/or change baud rates, codes, numbers of channels transmitted, and formats. Designing microprocessor-controlled serial interfaces would be reduced to wiring a parallel input connector and to programming a PROM (*P*rogrammable *R*ead *O*nly *M*emory). Changing the interface tailored for one instrument to that for another would require changing PROM's. Standard, high-volume serial interfaces between commercially available instruments and commercially available computers would require the same microprocessor-based interface and a much cheaper, mass-produced ROM (*R*ead *O*nly *M*emory).

In summary, serial computer interfaces can be made quite generally applicable for interfacing many instruments having parallel I/O with computers and many hard-copy devices. Serial interfaces suffer only when required to handle extremely high data rates. Finally, one should not underestimate the power, simplicity, and convenience of a single pair of wires for a computer interface using serial data exchange techniques.

References

(1) R. E. Dessy and J. Titus, *Anal. Chem.*, **46**, 294A (1974).
(2) J. Finkel, "Computer-Aided Experimentation: Interfacing Minicomputers", Wiley, New York, N.Y., 1975.
(3) D. G. Larsen and P. R. Rony, "The Bugbook IIA", E & L Instruments, Inc., Derby, Conn., 1975.
(4) D. G. Larsen and P. R. Rony, *Am. Lab.*, **7**, 116 (1975).
(5) "SERDEX User's Guide", Analog Devices, P.O. Box 280, Norwood, Mass. 02062.
(6) D. H. Lorimer and A. T. Bell, *Ind. Eng. Chem. Fundam.*, **15**, 71 (1976).