ISTITUTO NAZIONALE DI FISICA NUCLEARE

Sezione di Milano

INFN/BE-90/09
30 Luglio 1990

W. Bragagnini, P. Guazzoni, M. Pitalieri, G. Taiocchi and L. Zetta:

A SPECIAL PURPOSE PARALLEL-TO-SERIAL BIDIRECTIONAL INTERFACE

Servizio Documentazione
dei Laboratori Nazionali di Frascati

INFN - Istituto Nazionale di Fisica Nucleare
Sezione di Milano

INFN/BE-90/09
30 Luglio 1990

W. Bragagnini, P. Guazzoni, M. Pitalieri, G. Taiocchi, and L. Zetta

A special purpose parallel-to-serial bidirectional interface

A special purpose parallel-to-serial bidirectional interface

Walter Bragagnini[1,*], Paolo Guazzoni[1], Maurizio Pitalieri[1,+], Gianfranco Taiocchi[2], and Luisa Zetta[1]

(1) Istituto Nazionale di Fisica Nucleare and Dipartimento di Fisica dell'Universitá - Via Celoria 16 - 20133 Milano - Italy

(2) Ditta Takes - Ponteranica Bergamo - Italy

(*) present address: Aeritalia - Divisione Avionica - Ronchi dei Legionari (Go)

(+) present address: Harris - Scientific Calculations - Agrate B. (Mi)

ABSTRACT

This paper describes general structure and performance of a special purpose RS232 interface, designed for computer aided testing of particular devices, like, for instance, microprogrammed apparatuses or computational circuits.

1- INTRODUCTION

In order to perform computer aided testing of particular devices, like bit-slice microprogrammed units [1], or to connect special purpose computer aided pulse generators to the host computer [2,3], we have designed, built and tested a special purpose RS232 interface (Interf). It is able to connect, in simple way, the device under testing with the host computer (Host), assuring the proper data-transfer and handshake.
Moreover Interf is bidirectional, designed to transmit command words and data -from Host to the system under testing (System)-, and to receive -from the System- the results obtained and the status word, to be transmitted to Host.
In particular, any storage of data is made by Host, in order to make more simple Interf. In fact Interf acts only as a transmitter-receiver, giving the proper parallel-to-serial and serial-to-parallel conversions for data, results and command or status words, generating the required handshake signals.

2- GENERAL DESCRIPTION

The constraints to be satisfied in building Interf were:
- transmitting and receiving speed of 9600 Baud
- word length of 7-bit, even parity, 2 stop bit
- signals to be used, only TX and RX (serial-in and serial-out)
- other control signals (foreseen in RS232 standard) to be always active, in order to make the handshake easier and the two-way communication always possible. This simplification is allowed by the speed of circuitry under testing, always faster then RS232 hanshaking timing and by the bufferization, with latches, of the results in the circuit under testing.

A preliminary description of Interf can be given following fig.1, showing a simplified schematic. Two main blocks (transmitter and receiver) are clearly displayed, with the indication of all the signals, exchanged with the outside, together with their propagation way. Furthermore, each signal on the right represents the handshake with System, while the left ones that with Host.
Receiver accepts the serial data coming from Host, transforms them in parallel words of proper size (16/32-bit for data, 6/8-bit for commands and status words), sending them to System. Transmitter accepts the results, computed by System on the two data, together with command and status words, transforms them in serial data for Host, and sends also a control word to System (end of data, EOD).
The schematic of fig.2 allows to understand the working mode of Interf. It is based on an UART (Universal Asyncronous Transmitter/Receiver) -RCA CDP1854-, that allows a full programmability of data and control words, used for communication. Beyond allowing baud-rate and data setting, it signifies also transmission errors (e.g. parity). In fig.2 UART is drawn together with its internal schematic, in order to understand better the general

working mode. In particular, the drawn corresponds to 'zero mode', chosen for setting. So UART has a full compatibility with other transmitter/receiver circuits, and gives all signals needed for controlling the communication. It needs also two clock signals, one for transmission and the other for reception. In our case, receiving and transmitting speeds are equal. So just one clock signal is enough, which has to be 16-time the chosen baud-rate (9600). It is obtained with a NE555, used as oscillator at a frequency of 153.6 kHz, with an external potentiometer for frequency adjustment. The obtained stability is completely enough for good working of Interf. The control signals needed were set, in order to obtaine the liked communication parameters.

We can now come back to general working mode, in order to describe, always following fig.2, a typical operational sequence of Interf. Signals, coming from Host along RS232-line (Serial-in), pass trough an operational amplifier (TL084), used as comparator, to obtain right polarities. The same happens for Serial-out. In this case, for making possible connection of very long communication cables (up to hundreds of meters), a line-driver, two transistors push-pull connected, was employed.

UART reception period begins with recognizing start-bit at Serial-Data-in (SDI) input. After that, data, parity and stop bits are stored in the Receiver Shift Register (RSR). If programmed, as in our case, the parity bit and the reception of a valid stop bit are controlled. The received datum is loaded in Receiver Holding Register (RHR). If the word length is less than 8-bit, zeros are loaded in the MSBs not used. After RHR loading is completed, Parity Error and Framing Error signals are enabled for the charged character, together with Data Available signal (DA) used to signify to external circuitry that a good datum is available. This sequence is repeated for each received character. In coincidence with DA, the received datum is put out and sent to a Decoder used to recognize the data to be accepted.

In fact this special purpose interface was designed in order to simplify either the control program, or the handshake sections of the systems to be tested, or as well Interf itself. As consequence, it was decided to build Interf so as to be able to recognize only numbers (octal format, ASCII code) and some special characters, necessary for communication.

In ASCII code, in fact, the characters corresponding to octal values $0 \div 7$ have a binary configuration corresponding to that of numbers $0 \div 7$ with in addition the bits 5 and 6 always high, while the bits 4 and 7 are always low. Otherwise, no other character has such combination for bits 4,5,6,7. So, we can use the content of these four bits to distinguish octal numerical values (true data in our case) from the other words. In addition in this latter case, we can look for control characters used.

Coming back to flowing data, decoding is made by means of a proper diode matrix, acting on 7-bit input datum. In this way an octal number $0 \div 7$ consists always of a 3-bit word. The character identification is made on the basis of correspondences shown in Table 1.

In Table 2 are shown typical communication strings allowed by Interf, properly coded. In fact in order to write, using a Vax computer, a 32-bit number in "Interf octal standard" (11-digits), it is enough to write, in Fortran Language, a real number using octal format. Our representation of 11 octal number has MSB of MSD (most significant digit) always equal to zero.

In fig.3 the decoder-circuit for octal-number is shown. As it is possible to see, the operation is very simple, because of the strict transmission rules adopted. In fact it is enough to work on bits 4,5,6,7 in order to recognize an octal number. The number value, as previously said, is given by the first three bits, that are properly stored.

The flags, so enabled, are sent to a D-type flip-flop, to store its status. So, it is impossible to make wrong operations, otherwise possible also because UART is not strictly controlled due to the chosen philosophy. In fact Interf sincronization is made only by the recognized control characters.

Up to now, two important goals are achieved: a 'good' datum is accepted and subsequently identified { octal number (number-flag on) or control character (corresponding character-flags on) }, and transmitted words, differing from Table 1 codification, are ignored.

In order to reconstruct, as previously indicated, the structure of words to be sent to System, we chose a double parallel-to-serial and serial-to-parallel conversion. The three LSBs of the datum (in fact in our case, they represent the true octal value of the number) are stored in a register with a parallel input and serial output. The output of this register fills the input of a cascade of serial-parallel registers (74164). Their work is to reconstruct the 32-bit string (for data) or 6-bit word (for command codes) to be sent to testing System. As example, the schematic of the block to do this operation for data is shown in fig.4. The one for control codes is very similar. We have to remark that the clock signal of the serial-to-parallel and parallel-to-serial shift register for command codes comes from a multiplexer, drived by the command-flag.

Continuing the analysis of the working mode of Interf, we can now consider the timing of data transmitted from System to Host. In this case we have to solve a problem, that is fully symmetric if considered in comparison with the receiving problems. In fact, we have to send a serial bit-string, starting from the data received in parallel format. The data arrive to Interf, coming from System, as a 32-bit data-word (11 octal digits, MSB=0) or a 8-bit status-word (3 octal digits, MSB=0).

Data and status words are directly sent to inputs of multiplexers. They work as parallel-to-serial converters. As shown in fig.5, three different multiplexer circuits (74150 type) are employed, each with 16 inputs. The multiplexers, in order to select the data input to be put-out, are connected to four lines (A,B,C,D), the outputs of a binary counter. So is possible to start with the 1st input, successively to enable the 2nd one, the 3rd and so on up to the last.

The three multiplexer outputs are collected in parallel, enter a small circuit to generate control characters, which are added to the transmitting words { "_" at 12th and "(" at 16th positions }, in order to increase the transmission security, completely avoiding unespected mixing between data and status words. At the end they reach the proper UART inputs.

3- Performance

As previously said, Interf works directly connected to a serial port of a host computer (in our case a VAX 8600), with a Baud rate of 9600, 7-bit word, parity even, 2 stop-bits. The communication is made by means of proper codes, written in Fortran Language, using the QIO libraries [4] for driving the peripheral Interf. Obviously these codes will have also the possibility to read data needed for testing systems and to control the goodness of results. At the beginning Interf was subjected to usual electronic tests in order to verify the correspondence between design constraints and actual working mode. After this set of tests, it was controlled by means of itself, directly connected to the bidirectional auxiliarity port of a CIT220+ video terminal [5]. In this case, the internal clock (153.6 kHz) is used for the Autotest operation and the connector to System is directly connected to that from System, in order to allow the data coming from input to be put out at high frequency (fig.6). So connecting Interf to the bidirectional auxiliary port of CIT 220+, via RS232, it is possible to send data, by means of keyboard, and to show the same data, in reception, on video. This test allows a complete check up of only Interf, leaving apart any complication coming from connecting computers or systems to be tested.
Starting from the good working mode of Interf in Autotest, we employed it to test the Computational Logic Unit for the Microprogrammed Data Acquisition System of ref. 1 using VAX 8600 as host computer, and proper codes written, as previously said, using QIO libraries [5] for communication and exchange of data. The results achieved during several weeks were completely satisfactory.
A different kind of employment for Interf is two different types of generators, computer aided, as previously said. In these two cases Interf was built-in with a simplified structure, also because they need only the receiving part.
In the first case, the generator of ref. 2 is able to store 2 K of 12-bit word quadruplets in an internal memory, and to put out the quadruplets with variable frequency up to 700 kHz. The interface is used to receive the data from host computer, converted from serial to parallel before storage into memory. It was succesfully used also in the test of previous bit-slice apparatuses [6].
The generator of ref. 3 is able to convert four 12-bit digital words, coming from a data base stored in the host computer, via RS232 line and Interf, into the corresponding analog pulses, and to put them out with an amplitude ranging between $0 \div 4$V.
In all previously described cases, the reliable working mode of the systems, is a clear evidence of the goodness of the philosophy chosen in designing Interf and of technical solutions used.

References

[1] W. Bragagnini, P. Guazzoni, M. Pitalieri, G.F. Taiocchi, L. Zetta
Euromicro 90 - Amsterdam 1990, and to be published on Microprocessing and Microprogramming.

[2] M. Annunziata, W. Bragagnini, P. Guazzoni, G. Sechi, G.F. Taiocchi, L. Zetta
Nucl. Instr. and Meth. A271, 1988, 597

[3] S. Banfi, P. Guazzoni, G.F. Taiocchi, L. Zetta
Report INFN/BE - to be published

[4] DEC, VAX/VMS I/O User's Guide

[5] CIE TERMINALS - CIT 220+ User's manual

[6] M. Annunziata, P. Guazzoni, L. Lisca, G. Sechi, L. Zetta
Microprocessing and Microprogramming 18, 1986, 515

TABLE 1

	BIT NUMBER		
	H H H $\overline{4}$ 5 6 $\overline{7}$	——»	octal number, 'good number' (to/from host)
FROM HOST	$\overline{1}$ 2 $\overline{3}$ 4 $\overline{5}$ 6 $\overline{7}$	——»	"*" control character, General Reset
	$\overline{1}$ 2 3 4 5 $\overline{6}$ 7	——»	"^" control character, Kind Of Data (KOD)
	$\overline{1}$ 2 3 4 5 6 7	——»	"~" control character, Kind Of Data (KOD)
	1 2 3 4 $\overline{5}$ 6 $\overline{7}$	——»	"/" control character, End Of Data (EOD)
TO HOST	1 2 3 4 5 $\overline{6}$ 7	——»	"_" control character, End Of Result (EOR)
	$\overline{1}$ $\overline{2}$ $\overline{3}$ $\overline{4}$ $\overline{5}$ 6 $\overline{7}$	——»	"(" control character, End Of Status Word

TABLE 2

```
                        STRING FROM HOST
* 04 ^ 20000127635 00000040500/
|  |  |  |           |          |
|  |  |  |           |          └——> End Of Data
|  |  |  |           └——> 2nd datum    (= 3)
|  |  |  └——> 1st datum    (=  -7.1622708 E-11)
|  |  └——> data length  (32/16-bit)
|  └——> operation code (if required): division
└——> general reset

              2nd datum
Result =    ———————————
              1st datum

                        STRING  TO  HOST
01160351034 000(
 |         T |  └——> End Of Data
 |         | └——> status word (operation completed without errors)
 |         └——> End Of Result
 └——> result  (= -4.1886157 E10)
```

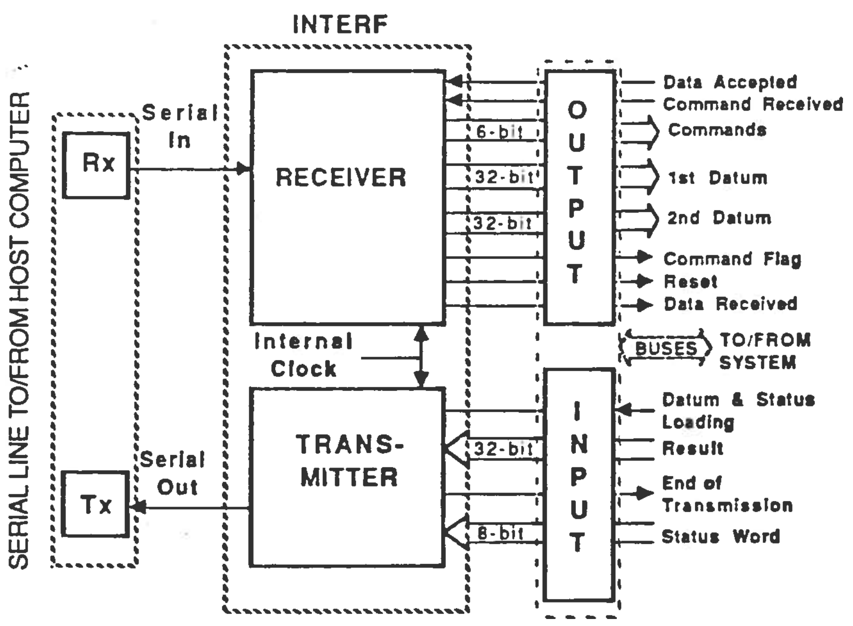

Fig. 1 - Simplified Schematic

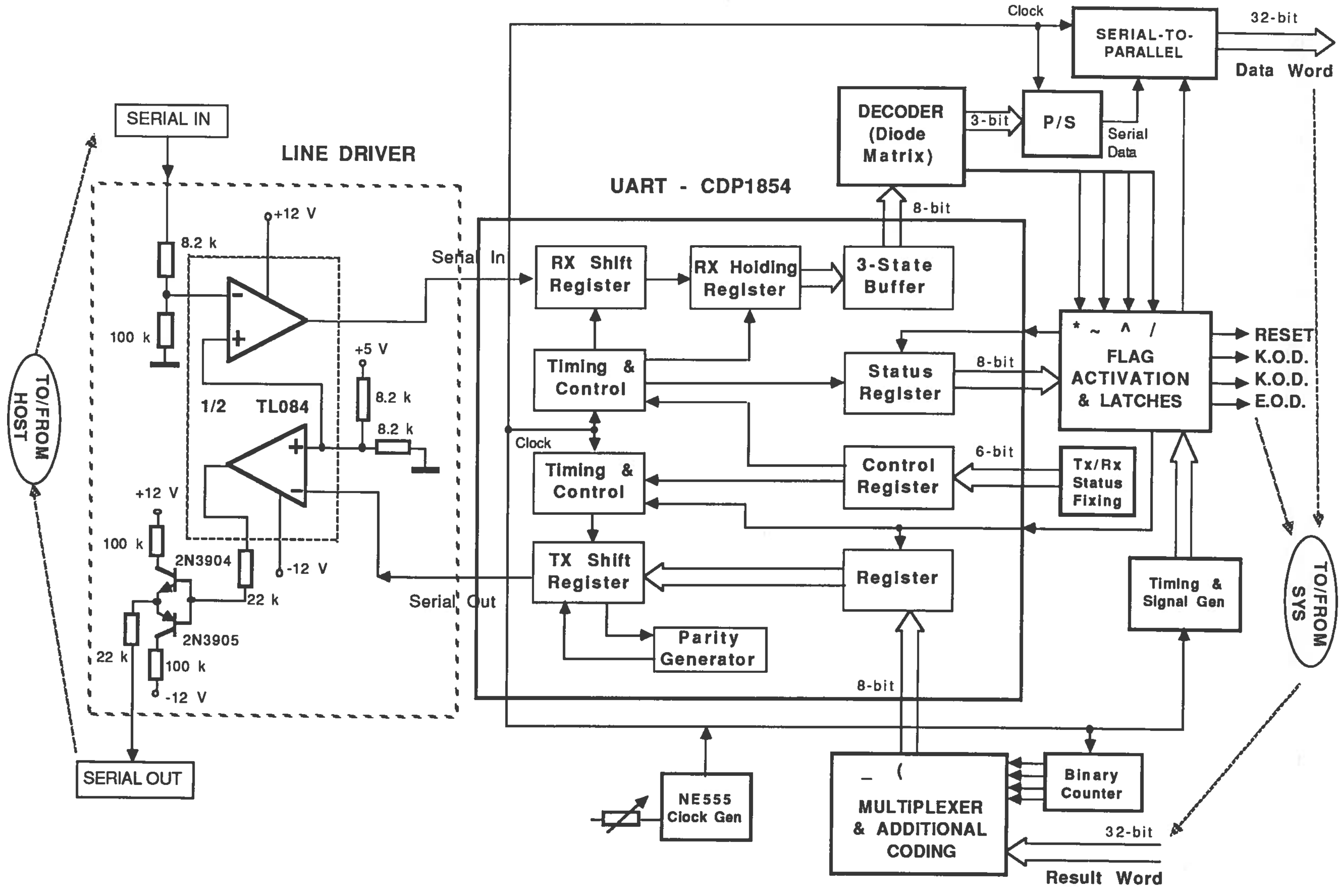

Fig. 2 - General Block Diagram

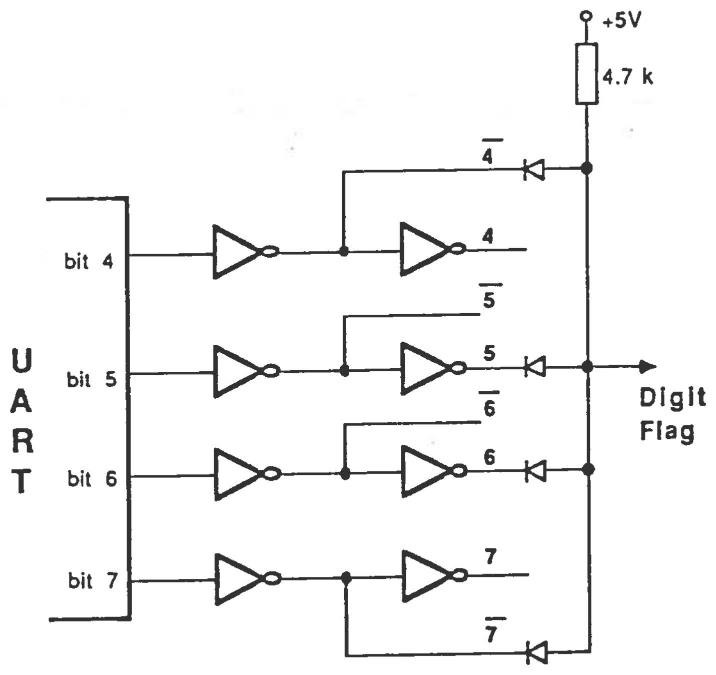

Fig. 3 - Schematic of the octal-number flag generator

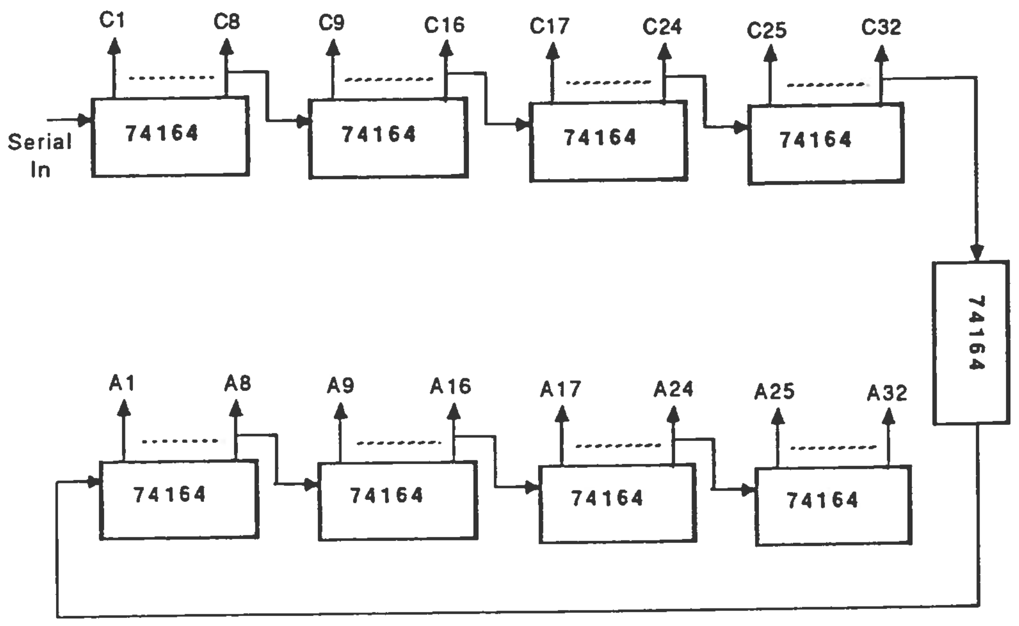

Fig. 4 - Schematic of Serial-to-Parallel converter

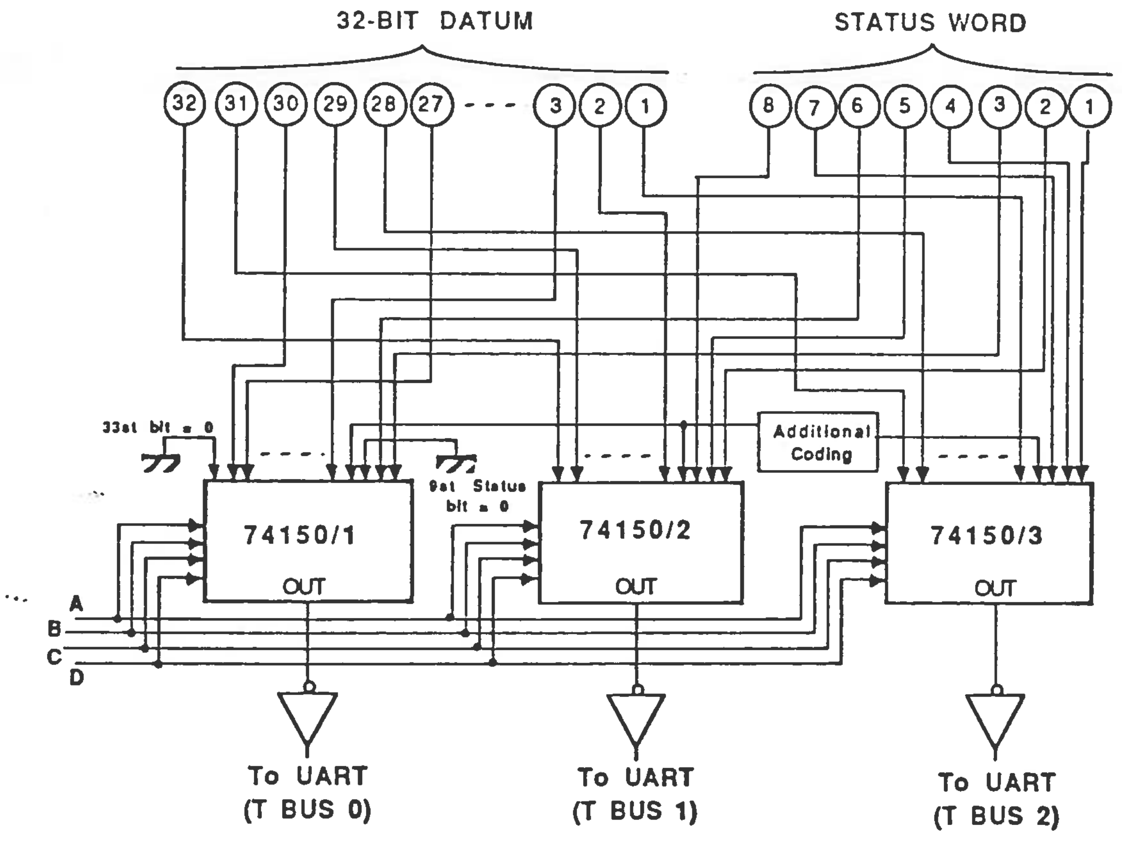

Fig. 5 - Schematic of Parallel-to-Serial converter

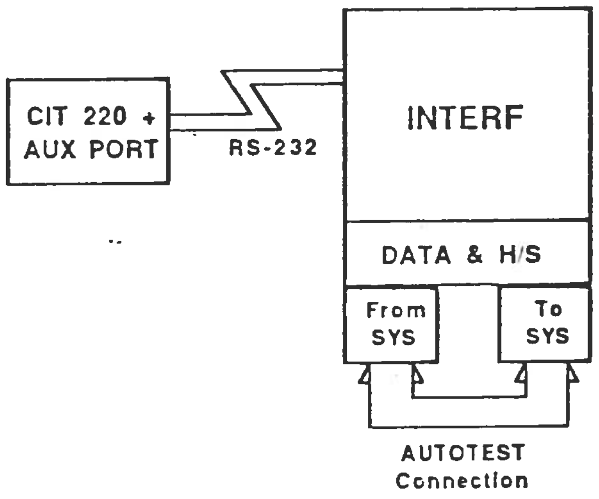

Fig. 6 - Autotest circuit block diagram