FORTRAN SIMULATION OF DIGITAL LOGIC

Dale I. Rummer
University of Kansas
Lawrence, Kansas

SUMMARY

The software described in this paper was created to aid engineering students verify the correctness of the design of digital logic systems. The verification consists of comparing a truth table generated by the simulation with the truth table desired. The software is written in FORTRAN because computers with FORTRAN compilers are widely available both to students and practicing engineers. The digital logic is defined in terms of actual hardware, rather than in terms of the classical AND, OR, and NOT functions. This feature makes it easy to write the simulation program directly from a hardware logic diagram. Thus the design is verified at the hardware level, rather than at the classical logic level. The implementation described is in terms of logic hardware produced by the Digital Equipment Corporation, but the hardware produced by any manufacturer can be described in similar terms. The method of simulation is applicable to a wide variety of both combinational and sequential logic. The restriction is that the operation of the logic system can be described by a sequence of decisions which are independent of subsequent decisions. For example it is not convenient to simulate the interconnection of two NAND gates for operation as a FLIP-FLOP. Other cases of feedback around several logic stages may also cause problems. This software is not concerned with aiding in minimization of the logic structure. The nature of the formulation of the simulation lends itself to automatic checking for fan-in or fan-out limitations, but this also is not the concern of this paper. The operation of the logic is assumed to be independent of propagation delays which there fore are not simulated. This method of simulation has been used to verify student designs for a pipeline control system, a degital message switching system, and a printer control system for a digital computer.

GENERAL IMPLEMENTATION

Although many FORTRAN compilers have the Boolean operators built in, these operators are not as convenient for the simulation of digital logic as are some special functions which can be easily defined. Fig. 1 shows three basic logic functions, their truth tables, and a FORTRAN arithmetic statement function which simulates this operation. All logic variables are assumed to have either the value zero or one. The correctness of these functions can be easily verified by direct substitution of the indicated values. Fig. 2 shows how the NAND and NOR functions are simulated in terms of previously defined logic functions. Although this approach may not yield minimum program length, nor minimum execution times, it does minimize the time, and hence, expense of software development. The user can always write the more complex logic function directly in terms of sums and products if desired. Fig. 3 shows additional examples of the derivation of three and four input logic functions in terms of the original two input functions.

SPECIFIC HARDWARE

Fig. 4 shows how a particular hardware device may be used to implement either a NOR function or a NAND function depending upon the definition of the logic as positive or negative. The method of simulation described will work for either convention, or for a mixed convention, just as hardware will function under any of these circumstances. For the sake of a consistent point of view in this paper, the following description will be in terms of negative logic. The logic function which simulates a particular item of hardware is given a name compounded from the manufacturer's part number and the lable of the output terminal for that particular function. This procedure makes it easy to compose the FORTRAN statements which constitute the simulation program directly from a hardware logic diagram. The possible errors introduced by translating back into classical logic functions are thus eliminated. This procedure is illustrated in Fig. 5 for the DEC type 107 **INVERTER** card which has seven inverters. The logic function associated with terminals d and e is called R107D since the output for that section appears on terminal d. This procedure of naming the FORTRAN statement functions which simulate a particular item of hardware is further illustrated in Fig. 6 for the R121 Quad NAND gate card.

EXAMPLE OF COMBINATIONAL LOGIC

Consider the simulation of a binary-to-octal converter as an example of how the FORTRAN simulation program would be written. Fig. 7 shows the truth table for this converter and Fig. 8 shows the logic block diagram on the right side. The three binary inputs to the converter are A, B, and C. Inverters are used to produce the inverse of each of these variables denoted as AB, BB, and CB. The output of the first inverter is AB and hence the FORTRAN replacement statement,

```
            AB = R107D( A ),
```

simulates the operation of this inverter. The remaining statements for BB, and CB are obtained in similar fashion. The R121L section of a NAND card is used to compute the variable XOB which is then inverted to obtain XO. The FORTRAN statements to simulate the operation of these two hardware devices are:

```
        XOB = R121L (AB,BB,CB)
         XO = R107L (XOB)
```

The sequence of these statements must be as shown to provide the proper value of the argument, XOB, for the inversion process. The other seven outputs are simulated in the same manner. The simulation of the operation of the binary-to-octal decoder consists of executing in sequence the series of statements shown on Fig. 8. It is assumed in this discussion that the values of the variables A, B, and C have been previously defined by computation, or by reading in as data. Since this sequence of statements must be executed once for each set of inputs, these statements might well be enclosed in a DO LOOP, or perhaps constructed as a subroutine. The same type of logic card may be used in several different locations in the hardware which is being simulated. Such a card needs to be defined by only one set of FORTRAN functions. The procedures indicated can be extended to simulate any combinational logic device which may be of interest.

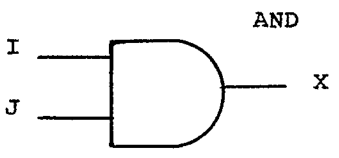
FLIP-FLOP CIRCUITS

The operation of a FLIP-FLOP can be simulated by appropriate combinational logic functions together with "memory" of the previous state. If this memory is provided within the function which defines the FLIP-FLOP, a separate function must be defined for each FLIP-FLOP which is needlessly wasteful of computer memory. Although FORTRAN functions are formally limited to one output value, extra input parameters can be used to provide the memory of the previous states which may be of interest. Fig. 9 shows the external connections for one FLIP-FLOP of the pair found on DEC type R205 cards. The conditions for setting, clearing, or complementing the FLIP-FLOP are also shown. The inputs F,D, and M are effective in altering the condition of the FLIP-FLOP only when there has been a change from the one state to the zero state. The normal state is the one state. The flow chart for a FORTRAN subprogram to simulate the operation of this FLIP-FLOP is shown in Fig.10. It is possible to define an arithmetic statement function to simulate the operation of a FLIP-FLOP. However, the mainline programming is made more tedious by the necessity of saving the previous value of pertinent variables. The use of a subprogram permits these operations to be coded just once, rather than once each time the FLIP-FLOP function is used. Other types of FLIP-FLOPS may also be simulated by similar definitions.

PARALLEL TO SERIAL CONVERTER EXAMPLE

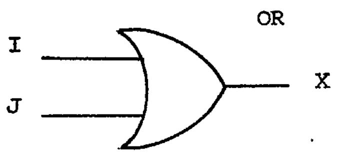
As an example of the simulation of both sequential and combinational logic, consider the five-bit parallel to series converter shown in Fig. 11. The input consists of logic levels present on the input lines P0 thru P4. After the register has been cleared by a CLEAR pulse, the inputs are transferred to the shift register by the LOAD pulse. The shift pulse gates the state of the upper FLIP-FLOP to the output bus, and causes the states of the FLIP-FLOPs to shift upward. The essential segments of the FORTRAN program to simulate the operation of this shift register in the context of a FORTRAN subroutine are shown in Fig. 12. Fig. 13 shows a print-out such as would be produced by the computer simulation of the parallel to serial converter. The register was assumed to be all ones initially to show the effect of the clear pulse. The transition of the LOAD bus from the one state to the zero state causes the data on the INPUT buses to be transferred to the shift register. The transitions from the one state to the zero state of the SHIFT bus causes the data in the shift register to be shifted up one stage. The serial sequence of ones and zeroes on the OUTPUT bus corresponds to the original data in parallel form on the INPUT buses. The OUTPUT bus returns to zero each time the SHIFT bus goes to zero. This point is made clear by the second case where the parallel data consists of all ones. Note that this new data appeared on the INPUT buses several shift cycles prior to the second LOAD pulse.

LOGIC SYMBOL | TRUTH TABLE | FORTRAN STATEMENT FUNCTION



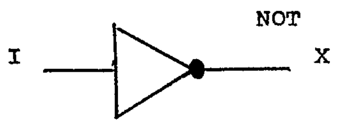
I	J	X
0	0	0
0	1	0
1	0	0
1	1	1

AND2(I, J) = I * J



I	J	X
0	0	0
0	1	1
1	0	1
1	1	1

OR2(I, J) = I+J -I*J

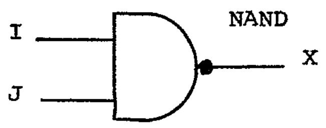


I	X
0	1
1	0

NOT(I) = 1 - I

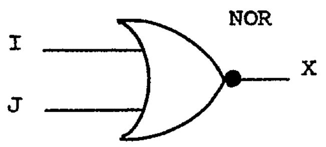
Fig.1 : BASIC LOGIC FUNCTIONS AND THE RELATED FORTRAN SIMULATION

LOGIC SYMBOL | TRUTH TABLE | FORTRAN STATEMENT FUNCTION



I	J	X
0	0	1
0	1	1
1	0	1
1	1	0

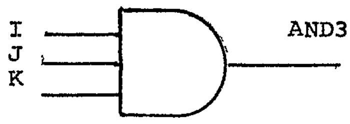
NAND2(I,J) =
NOT(AND2(I,J))

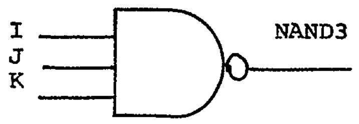


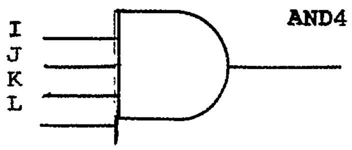
I	J	X
0	0	1
0	1	0
1	0	0
1	1	0

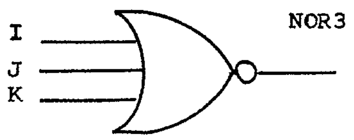
NOR2(I,J) =
NOT(OR2(I,J))

Fig.2 : LOGIC FUNCTIONS DERIVED FROM THE BASIC LOGIC FUNCTIONS





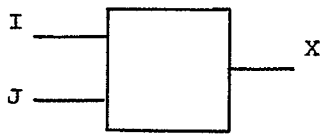




```
AND3( I,J,K )
   = AND2 ( AND2( I,J), K )

AND4( I,J,K,L )
   = AND2 ( AND2( I,J), AND2( K,L )

NAND3( I,J,K )
   = NOT( AND3( I,J,K ) )
NOR3( I,J,K )
   = NOT( OR3( I,J,K ) )
```

Fig.3 : ADDITIONAL DERIVED LOGIC FUNCTIONS

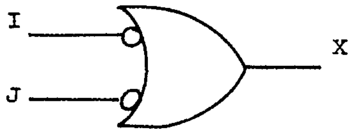
HARDWARE



H Higher level
L Lower level

I	J	X
H	H	L
H	L	L
L	H	L
L	L	H

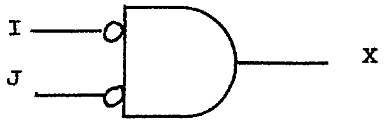
POSITIVE LOGIC



H = 1
L = 0

I	J	X
1	1	0
1	0	0
0	1	0
0	0	1

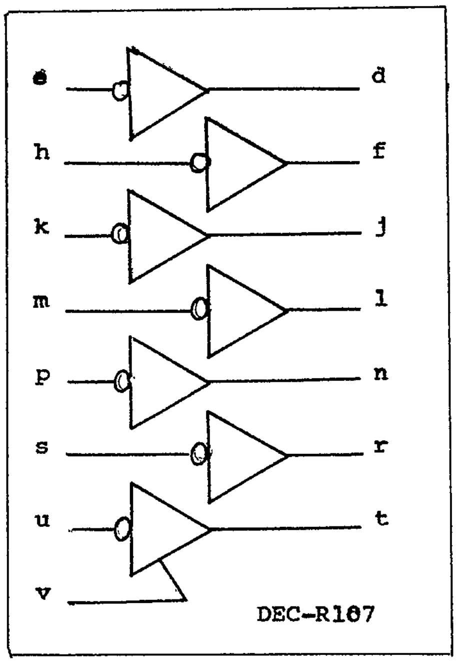
NEGATIVE LOGIC



H = 0
L = 1

I	J	X
0	0	1
0	1	1
1	0	1
1	1	0

Fig. 4: POSITIVE AND NEGATIVE LOGIC CONVENTIONS

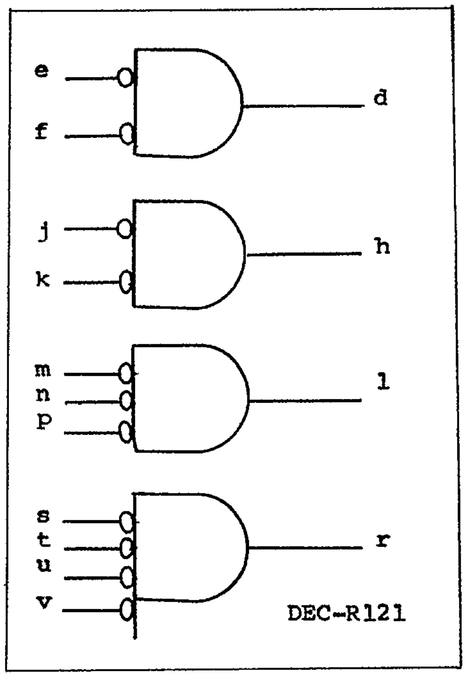


FORTRAN STATEMENT FUNCTIONS

```
R107D( I)  =  NOT( I )

R107F( I)  =  NOT( I )

R107J( I)  =  NOT( I )

R107L( I)  =  NOT( I )

R107N( I)  =  NOT( I )

R107R( I)  =  NOT( I )

R107T( I)  =  NANDn( I, .....)
```

note: terminal v permits use of external diode networks.

Fig. 5 : DEC TYPE R107 INVERTER LOGIC CARD

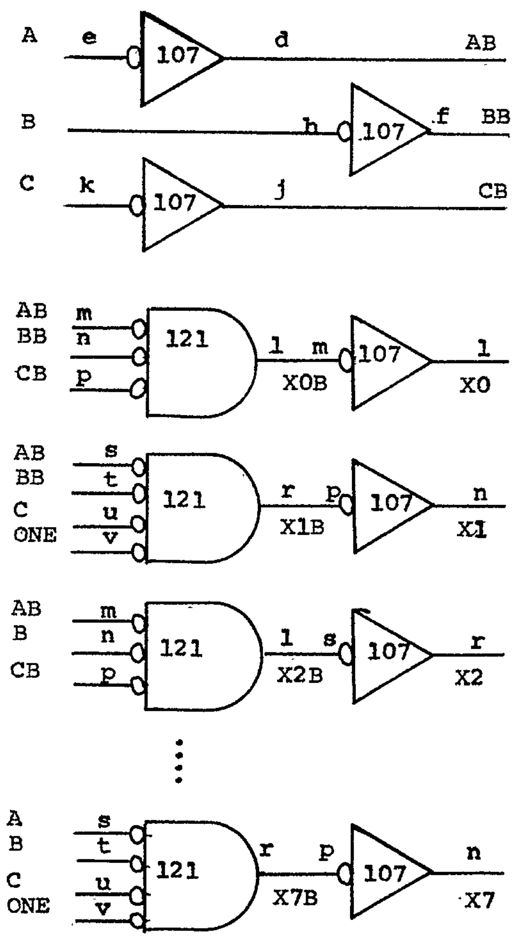


```
R121D( I,J)  =  NAND2( I,J)

R121H( I,J)  =  NAND2( I,J)

R121L( I,J)  =  NAND3( I,J,K)

R121R( I,J,K,L) =  NAND4( I,J,K,L)
```

Fig.6 : DEC TYPE R121 NAND GATE LOGIC CARD

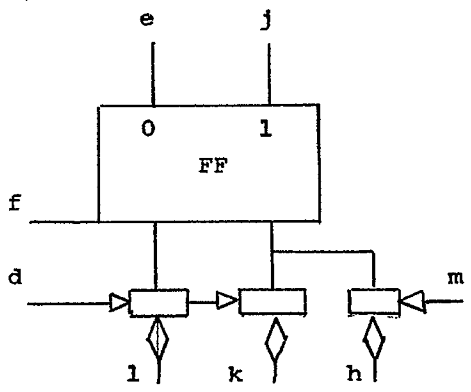
A	B	C	X0	X1	X2	X3	X4	X5	X6	X7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Fig. 7 : TRUTH TABLE FOR BINARY TO OCTAL CONVERTER



```
AB  =  R107D( A )

BB  =  R107F( B )

CB  =  R107J( C )


XOB = R121L( AB,BB,CB )
X0  = R107L( XOB )

X1B = R121R( AB,BB,C, ONE )
X1  = R107N( X1B )

X2B = R121L( AB,B, CB )
X2  = R107R( X2B )


X7B = R121R( A, B, C, ONE )
X7  = R107N( X7B )
```

Fig. 8 : LOGIC CIRCUITS AND FORTRAN CODING FOR SIMULATION OF A BINARY TO OCTAL DECODER USING DEC LOGIC CARDS



F	D	M	L	K	H	CLEAR	SET	COMP.
T*	1	1	1	1	1	1	0	0
1	T	1	0	1	1	1	0	0
1	T	1	0	0	1	1	1	1
1	T	1	1	0	1	0	1	0
1	1	T	1	1	0	0	0	1

* a transition from <u>one</u> to <u>zero</u>

Fig. 9: CONNECTIONS AND OPERATION OF THE FLIP-FLOP FOUND ON ONE HALF OF DEC TYPE R205 CARD

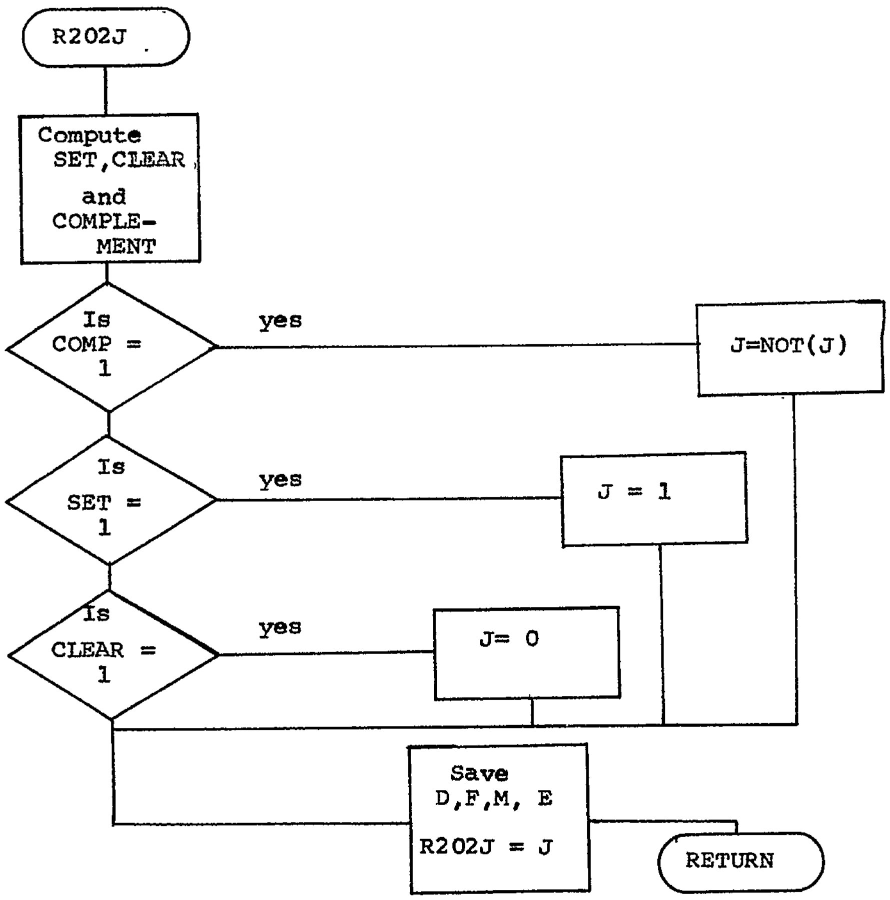


Fig. 10: FLOW CHART FOR FLIP-FLOP SIMULATION SUBPROGRAM

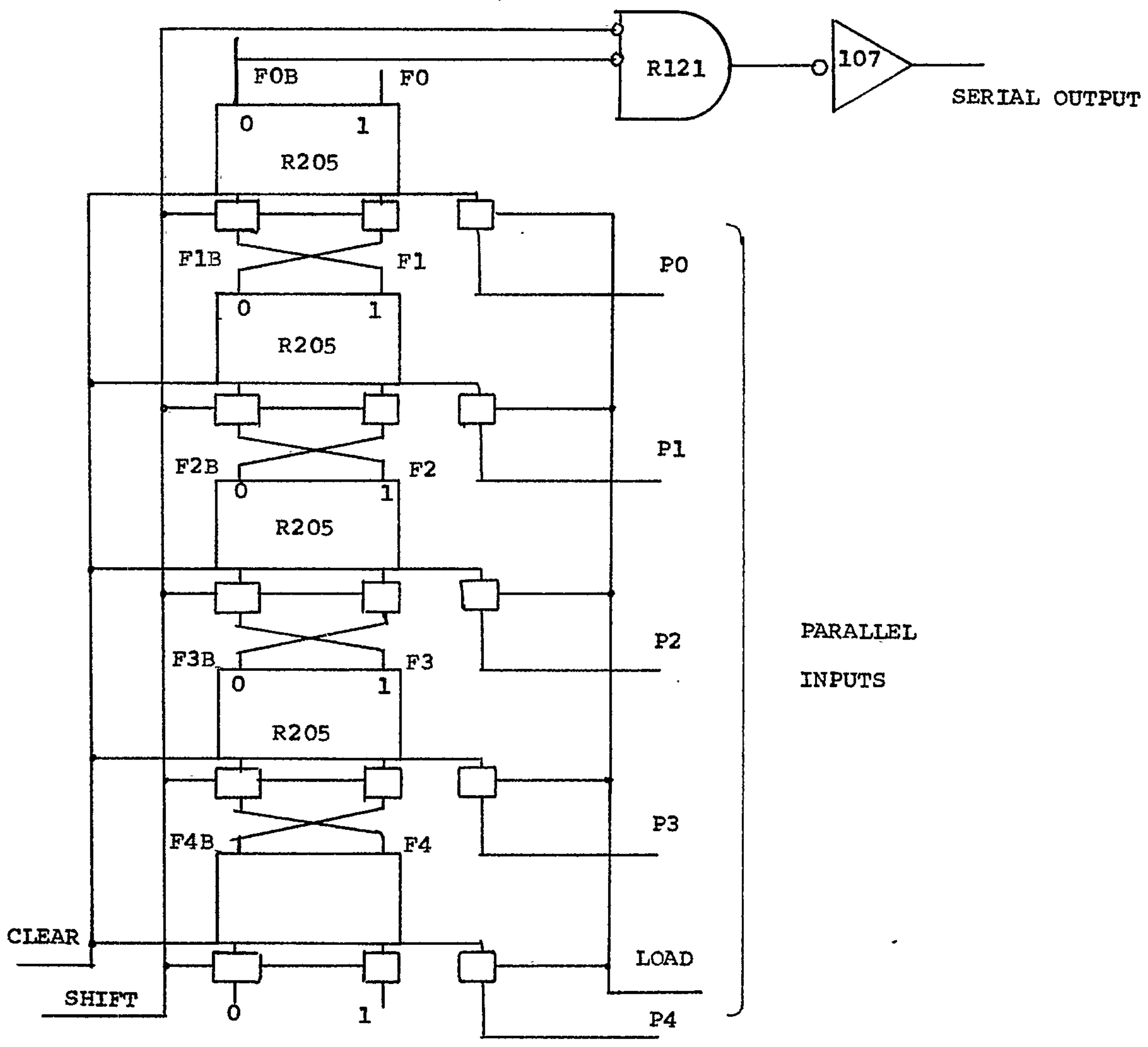


Fig. 11: LOGIC SCHEMATIC FOR FIVE-BIT PARALLEL TO SERIAL CONVERTER

```
SUBROUTINE  SHFRGR
COMMON ..............
DIMENSION CLEAR(2), SHIFT(2), LOAD(2) ........
.
.
F4 = R205J( CLEAR,SHIFT, 0, 1, P4, LOAD, F4B )
F3 = R205P( CLEAR,SHIFT, F4,F4B,P3,LOAD, F3B )
F2 = R205J( CLEAR,SHIFT, F3,F3B,P2,LOAD, F2B )
F1 = R205P( CLEAR,SHIFT, F2,F2B,P1,LOAD, F1B )
F0 = R205J( CLEAR,SHIFT, F1,F1B,P0,LOAD, F0B )
RETURN
END
```

Fig. 12: ESSENTIAL STATEMENTS FOR FORTRAN SUBROUTINE TO SIMULATE THE FIVE-BIT SHIFT REGISTER

```
CLEAR              PARALLEL                  SHIFT                     SERIAL
*   LOAD           INPUTS  P                 REGISTER F                OUTPUT
*   *   SHIFT      4   3   2   1   0         4   3   2   1   0         *
*   *   *          *   *   *   *   *         *   *   *   *   *         *
*   *   *          *   *   *   *   *         *   *   *   *   *         *
1   1   1          1   0   1   0   1         1   1   1   1   1         1

0   1   1          1   0   1   0   1         0   0   0   0   0         0

1   0   1          1   0   1   0   1         1   0   1   0   1         1
1   1   0          1   1   1   1   1         0   1   0   1   0         0
1   1   1          1   1   1   1   1         0   1   0   1   0         0
1   1   0          1   1   1   1   1         0   0   1   0   1         0
1   1   1          1   1   1   1   1         0   0   1   0   1         1
1   1   0          1   1   1   1   1         0   0   0   1   0         0
1   1   1          1   1   1   1   1         0   0   0   1   0         0
1   1   0          1   1   1   1   1         0   0   0   0   1         0
1   1   1          1   1   1   1   1         0   0   0   0   1         1
1   1   0          1   1   1   1   1         0   0   0   0   0         0
1   0   1          1   1   1   1   1         1   1   1   1   1         1
1   1   0          1   1   1   1   1         0   1   1   1   1         0
1   1   1          1   1   1   1   1         0   1   1   1   1         1
1   1   0          1   1   1   1   1         0   0   1   1   1         0
1   1   1          1   1   1   1   1         0   0   1   1   1         1
1   1   0          1   1   1   1   1         0   0   0   1   1         0
1   1   1          1   1   1   1   1         0   0   0   1   1         1
1   1   0          1   1   1   1   1         0   0   0   0   1         0
1   1   1          1   1   1   1   1         0   0   0   0   1         1
1   1   0          1   1   1   1   1         0   0   0   0   0         0
```

Fig. 13: COMPUTER LISTING FROM SIMULATION OF FIVE BIT PARALLEL TO SERIAL CONVERTER