nectivity consideration detects and locates logical errors in the information flow. Similar techniques can be used in the debugging and maintenance of computer programs.

Much fruitful work remains to be done. Fast computer algorithms are needed to manipulate large graphs. Perhaps one can specify various degrees of connectivity. The basic methods of activating test inputs and monitoring the test outputs must be investigated.

ACKNOWLEDGMENT

The author acknowledges his indebtedness to his many colleagues at Honeywell for suggestions and discussions. He is grateful to Dr. R. B. Lawrance for his continued encouragement.

APPENDIX I—A modified method of selecting prime Implicants

1. Given the test point range matrix $M$, compute the row weights (number of ones in a row) and the column weights of the matrix.

2. Find the row with the least weight. Select the column with the largest weight which is implied by this row. Delete all rows subsumed by this implicant. Repeat Step 2 until no rows remain.

3. The selected columns correspond to the test points that are needed for almost minimal number of test point pairs.

For many simple cases, this algorithm provides optimal test point pair allocation. It is possible that variations of the above selection procedure can be found for improving the number of computations or closeness to optimality.

REFERENCES

1. R. ELDRED
   Test routines based on symbolic logic statements
   Journal of ACM Jan. 1959

2. W. KAUTF
   Automatic fault detection in combinational switching networks
   Proc. Symposium of Switching Theory and Logic Design 1961

   Techniques for the diagnosis of switching circuit failures
   Trans. IEEE Communications and Electronics Sept. 1964

   Design of serviceability features for IBM System 360
   IBM Journal of Research & Development April 1964

5. S. SESHU, D. FREEMAN
   Diagnosis of asynchronous sequential switching systems
   IEEE Trans.-EC Aug. 1962

   A self-diagnosable computer
   Proc. Fall Joint Computer Conference 1965

7. C. V. RAMAMOORTHY
   Analysis of graphs by connectivity considerations
   Journal ACM April 1966

8. C. V. RAMAMOORTHY
   The analytic design of a dynamic look-ahead and program segmentation system for multi-programmed computers

9. C. V. RAMAMOORTHY
   Generating functions of abstract graphs with systems applications
   Ph.D. Thesis Harvard University May 1964

10. E. J. McCLUSKEY
    Minimization of Boolean functions
    B.S.J.T. November 1956

11. R. A. JOHNSON
    An information theory approach to diagnosis
    Proc. Sixth National Symposium on Reliability and Quality Control Jan. 1960

References 12 and 13 are excellent summaries on related topics.

12. R. WARD and T. O. HOLTEY
    The maintainability factor in the design of digital systems using microelectronics
    Proc. WESCON 1966

    Structural models
    John Wiley & Sons New York 1965
Compiler level simulation of edge sensitive flip-flops

by JAMES T. CAIN and MARLIN H. MICKLE
University of Pittsburgh
Pittsburgh, Pennsylvania
and
LAWRENCE P. MCNAMEE
University of California
Los Angeles, California

INTRODUCTION

Recently there has been increased activity in the area of compiler level language simulation of digital systems. The compiler level language has a distinct advantage in that it places a valuable tool in the hands of the design engineer who usually has at least a prima facie knowledge of such a language.

R. P. Larsen, M. M. Mano,¹ and R. M. McClure² have recognized the need for some sort of low cost digital simulator to be available to all digital design engineers. They state that trends indicate more and more engineers are being confronted with digital design problems which are amenable to simulation techniques. However, many of these engineers are unable to use the powerful tool of simulation simply because the small size of the systems, or the fact that they are one or two-of-a-kind systems, does not justify the expense of developing a simulation program.

McClure and other individuals at Texas Instruments Inc. have attempted to fill the need for a low cost digital simulator by developing a special programming language for simulating digital systems.

Larsen and Mano have proposed using the FORTRAN IV standard programming language as a digital system simulation medium. The FORTRAN IV language was chosen because the logical Boolean operations are included in the instruction repertoire. They state that the simulation programming can be simplified through a systematic approach. In addition the digital network should be reduced to a set of the simplest possible digital elements (ANDS, ORS, NOTS, FLIP-FLOPS, etc.) and that the elements with memory (FLIP-FLOPS) should be simulated by a Boolean statement relating the element’s new output in terms of the present input and output values. For example, the mapping relationship that they suggest for a trigger flip-flop is:

\[ f(t_2) = s(t_1) + f(t_1) t(t_1) + r(t_1) t(t_1) f(t_1) \]  

where \( f \) is the assertion state of the flip-flop, \( s \), the set signal, \( r \), the reset signal, and \( t \), the trigger signal. The letters in parentheses, \( t_1 \) and \( t_2 \), refer to the present and new states respectively. This is a mapping relationship not an expression of the internal hardware configuration of the trigger flip-flop.

An inspection of the above relationship reveals that the mapping is level sensitive, i.e., it corresponds to a Boolean function. In simulation, a flip-flop is used considerably as a module in the building of ring counters, binary counters, etc. The counters which are built from these flip-flops depends on the edge-sensitive properties of the modules. In order to provide the simulation of devices such as a ring counter, it will be necessary to arrive at a mapping “relationship” in terms of a compiler level statement or statements which demonstrate the edge-sensitive properties of the flip-flop module. The purpose of this paper is to provide a compiler level subroutine which in itself is edge-sensitive. For the purpose of demonstration, the ring counter of Figure 1 will be used as a vehicle.

**Simulation**

The MAD (Michigan Algorithmic Decoder) language was chosen because (1) the system can compile and execute programs in the Boolean mode (i.e., Boolean equations can be written and input directly to the compiler), (2) the MAD language, like FORTRAN IV, can easily be available to digital design engineers, and (3) the language has powerful simplified input and output statements which augment the power of the language as a digital simulator.

A close inspection of the ring counter circuit of Figure 1 reveals that the S-R flip-flops used in its construction must be sensitive to the trailing edge of the set or reset signal. In simulating the flip-flops needed for this ring counter, two problems are encountered. First, a flip-flop is a memory element which means that feedback of signals is incorporated in its
operation. This feedback action in an actual flip-flop takes place in the parallel mode, but in the simulator, all operations must occur in the serial mode. Second, the set and reset terminals of the actual flip-flops are sensitive to the trailing edge of the signals so that this type of operation must be included in the simulation program.

The simulation of the trailing edge sensitive flip-flops in the ring counter was accomplished by using the external function capability of the MAD language. An external function is actually a subroutine which is compiled independently of the main MAD program. The external function for trailing edge sensitive flip-flops is shown in the following external function. The statements are numbered in the left hand margin for reference purposes.

**EXTERNAL FUNCTIONS FOR TRAILING EDGE SENSITIVE S-R FLIP-FLOPS**

1. EXTERNAL FUNCTION (S,R,A)
2. NORMAL MODE IS BOOLEAN
3. INTEGER I, II, IN, J, K, L
4. INTEGER FAKOI
5. DIMENSION SS (9), RR(9)
6. PROGRAM COMMON FAKOI, FAKO2
7. ENTRY TO TFFSR.
8. FAKOI = FAKOI + 1
9. WHENEVER FAKOI.E.1
10. K = 5
11. L = 2*K-I
12. II = K-I
13. J = II
14. THROUGH INIT2, FOR IN=0,1,IN
15. SS(IN) = OB
16. RR (IN) = OB
17. INIT2 CONTINUE
18. OTHERWISE
19. CONTINUE
20. END OF CONDITIONAL
21. WHENEVER J .E. I
22. J = K
23. THROUGH TR, FOR I = 0,1,II .G.II
24. SS(I) = SS(I+K)
25. RR(I) = RR(I+K)
26. TR CONTINUE
27. OTHERWISE
28. J = J + 1
29. END OF CONDITIONAL
30. SS(J) = S
31. RR(J) = R
32. WHENEVER .NOT.SS(J).AND.RR(J-K)
33. SSS = 1B
34. OTHERWISE
35. SSS = OB
36. END OF CONDITIONAL
37. WHENEVER .NOT.RR(J).AND.RR(J-K)
38. RRR = 1B
39. OTHERWISE
40. RRR = OB
41. END OF CONDITIONAL
42. FUNCTION RETURN SSS
43. END OF FUNCTION

Statements 1 through 6 are defining control statements. The variables S, R, and A, in statement 1 are dummy variables. When the subroutine is called, the actual program variables, which are in corresponding positions in the calling statement, are carried into the subroutine via these dummy variables. For this particular subroutine the dummy variable S is the set signal, R is the reset signal, and A is the output signal of the assertion side of the flip-flop. Statements 2 through 6 provide information for the compiler. These statements are necessary because the external function is treated as a separate program by the compiler.

The actual entry point to the subroutine, when it is called by the main program, is statement 7. Statements 8 and 9 are control statements which provide a means of initializing the subroutine when it is called for the first time. Statements 10 through 20 are the initialization statements. Statement 10 indicates to the system how many flip-flops (K=number of flip-flops) are to be simulated and the remaining initialization statements are a function of this statement.

The portion of the subroutine between statements 21 and 29 is designed to save core space. Since five flip-flops are being simulated, only five set and reset
signals need be saved. This enables the previous signal to be compared with the new signal to determine if it is the trailing edge of the signal. After all five flip-flops have been evaluated, these statements save the last five set and reset signals in SS (0) ... SS (4) and RR (0) ... RR (4).

Statements 30 and 31 transfer the current value of the set and reset signals, carried by the dummy variables, to the linear array. Statements 32 through 36 compare the current value of the set signal with the previous value for that particular flip-flop. If the signal has changed from high to ground (1 to 0), the trailing edge of the pulse is present. Therefore statement 33 sets the effective set signal to one. Otherwise the effective set signal is set to zero. Statements 37 through 41 perform the same function for the reset signals. After the program has evaluated the effective set and reset signals, the new value of the assertion terminal of the flip-flop can be evaluated by a mapping relationship between the input and previous output signals. Statement 42 evaluates this mapping relationship and returns to the calling program with the new value of the flip-flop output. Statement 43 serves only to tell the compiler where the external function definition terminates.

The following MAD program demonstrates how the external function is used in the simulation of the ring counter.

RING COUNTER

THROUGH END, FOR I == 1, 1, I.G.50
CLOCK = .NOT. CLOCK
SONE = CLOCK .AND.ZERO
ROKE = CLOCK .AND.ONE
ONE = TFFSR. (SONE, RONE, ONE)
STWO = CLOCK .AND. ONE
RTWO = CLOCK .AND. TWO
TWO = TFFSR. (STWO, RTWO, TWO)
STHR = CLOCK .AND. TWO
RTHR = CLOCK .AND. THREE
THREE = TFFSR. (STHR, RTHR, THREE)
SFOR = CLOCK .ANDR. THREE
RFOR = CLOCK .AND. FOUR
FOUR = TFFSR. CSFOR, RFOR, FOUR)
SFVE = CLOCK .AND. FOUR
RVE = CLOCK .AND. FIVE
FIVE = TFFSR. (SFVE, RFVE, FIVE)
END ZERO = .NOT.ONE .AND. .NOT.TWO .AND
.NOT.THREE. .AND. NOT.FOUR. .AND.
.NOT.FIVE

It is important to note that, by using this subroutine, statements were written which have a one to one correspondence to the actual hardware configuration of Figure 1. The group of statements which simulate the individual flip-flops must be ordered as shown because the set and reset signal of the next flip-flop depend upon the state of the previous flip-flop. In order to demonstrate the action in the ring counter, a MAD through loop was used. By the use of a through loop, an input clock signal is generated by a single statement, i.e.,

CLOCK = .NOT. CLOCK

The external function as presented is sensitive to the trailing edge of the incoming waveform. The function can be easily modified in order to sense the leading edge. The modification is performed simply by rewriting statements 32 and 37 of the external function as shown below.

32 WHENEVER .NOT.SS(J-K).AND.SS(J)
37 WHENEVER .NOT.RR(J-K).AND.RR(J)

SUMMARY

The authors believe the contribution of this paper is the development of the external functions or subroutines to simulate the edge sensitive devices which are an integral part of a digital system. The development of this and other subroutines enhances the power of the MAD language for use by non-programming oriented, digital design engineers. This is because the use of the subroutines produces a one-to-one correspondence between every logic gate in the simulated system and every MAD logic statement or subroutine call in the simulation program. This enables the design engineer to use the logic schematics of the computer system as flow charts for writing his simulation program.

The subroutine was constructed to be as flexible as possible. The external function capability of the MAD language was used instead of the internal function capability because the Michigan system treats an external function as a completely separate program. This means that a variable in an external function can have the same name as a variable in the main program or another external function and no multiple definition problems will result. This feature augments the flexibility of the subroutine which was developed in this paper.

The subroutines presented can easily be adapted to simulate any number of trailing edge sensitive flip-flops. A similar subroutine has been developed to simulate leading and trailing edge sensitive trigger flip-flops.

REFERENCES

1 R P Larsen MM Mano
   Modeling and simulation of digital networks
   Communications of the ACM 8 308-312 (May 1965)
2 R M McClure
   A programming language for simulating digital systems
   Journal of the Association for Computing Machinery 12
   14-22 (January 1965).