

AD-A090 641

BOEING AEROSPACE CO SEATTLE WA ENGINEERING TECHNOLOGY DIV F/B 16/4
NEW TECHNIQUES FOR TEST DEVELOPMENT FOR TACTICAL AUTOPILOTS USI--ETC(U)
JUL 80 E H SHEMETA

UNCLASSIFIED

NL

1 of 1
AD
A090 641

END
DATE
FILMED
11 80
DTIC

AD A090641

DDC FILE COPY

6

NEW TECHNIQUES FOR TEST DEVELOPMENT FOR TACTICAL AUTOPILOTS USING MICROPROCESSORS

10 JUL 1980

Edward H. Shemeta
Engineering Technology Dept.
Boeing Aerospace Company
Seattle, Washington 98124

Accession For	
DTIC GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By <i>Per M. on File</i>	
Distribution/	
Availability Codes	
Avail and/or	Special
Dist	
A	

SUMMARY

12/28

Advances in semiconductor technology have made it possible to introduce increasingly complex digital logic in avionics, weapons and control systems. However, the testing of such systems represents a new problem. Test requirements of missile systems are now approaching the complexity of aircraft themselves a decade ago. The costs of creating test procedures for the new designs will be prohibitive if new methods of test generation are not developed.

In the early 1970's, simulation methods were developed to generate test patterns on computers to increase productivity. These methods, e.g., 'LASAR', 'FAIRTEST', etc. were used successfully to support the A7-D/E and the S3-A aircraft test development needs. These methods have been also used to develop tests for E3-A, B-1 Avionics and IUS programs at Boeing.

As useful as these methods are, the computer costs for test generation are becoming high. Also, the time required to create computer models needed to support the test generation process itself has become very long. It has been estimated that to develop gate-level models for off the shelf LSI (Large Scale Integration) microprocessors and similar devices would take a man-year. Since the development phases of such programs are a year or less, the methods which worked so well in the past are not keeping up with the new requirements.

At Boeing, research in the test development area has been directed at developing new approaches that are applicable to the new technology. The most promising approach is based on the use of algorithmic test generation techniques instead of the classic truth table methods.

Using algorithms as the source for test code, it is not necessary to store millions of test vectors in a tester or to run costly simulations to generate the test vectors.

This paper reports on a demonstration of the application of the method to generate system level tests for a typical tactical missile autopilot. The test algorithms are based on the autopilot control law. When loaded on the tester with appropriate control information, the

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

Page 1

20 10 10 003
410975 AM



LEVEL II

DTIC ELECTE
S OCT 20 1980 D

complete autopilot is tested to establish if the specified control law requirements are met. Thus, the test procedure not only checks to see if the hardware is functional, but also checks the operational software. The technique also uses a 'learning' mode to allow minor timing or functional deviations from the expected responses to be incorporated in the test procedures.

A potential application of this test development technique is the extraction of production test data for the various subassemblies. The technique will 'learn' the input-output patterns forming the basis for development and production tests. If successful these new techniques should allow the test development process to keep pace with semiconductor progress.

INTRODUCTION

Technology advances in semiconductor design and manufacturing techniques have made the miniaturization of high performance systems a practical and economical concept. The replacement of marginal and sensitive analog hardware, such as servomechanisms and potentiometers in computing circuits, with digital processing circuits has impacted all aspects of electronic products. The result is the emergence of a whole new generation of 'smart' instruments. The commercial success of electronic games, etc., is a testimonial to the cost-effectiveness and versatility of the new digital techniques.

The application of this technology to low cost digital autopilots for small, highly accurate, terminally-guided missiles is particularly advantageous. A reliable low cost 'smart' missile which is field adaptable to a broad range of threats could be the ultimate next generation field weapon.

However, it is not enough for a system to be constructed from low cost parts and be functionally versatile. The system must be manufacturable, to assure low cost, and, more importantly, field supportable to achieve its potential as a weapon.

Unfortunately, systems based on microprocessor technology present new problems in developing effective production test procedures using traditional methods. The problem lies in the very advantages of such systems, i.e., the ability to change the system programming to meet new requirements without changing the hardware. As the software changes are made, the test procedures must likewise be modified. It is imperative that the system test procedures be capable of being modified as rapidly and effectively as the microprocessor software.

In this paper the traditional test approaches are reviewed as to their adequacy in meeting the new requirements, and a new approach, algorithmic test generation, will be outlined. This approach shows promise in answering these new requirements. An example of algorithmic test generation will be developed to illustrate the development of test algorithms from system level design data.

STATE OF THE ART IN DIGITAL TEST DEVELOPMENT

Early test generation methods in the vacuum tube era were based on signal tracing techniques using scopes and waveform diagrams. Standard input signals were generated and data flow was manually traced from input to output. With good access to the vacuum tube sockets, the test procedures were easy to develop and use.

With the advent of integrated circuits, the simple signal tracing concepts were more difficult to document and the inability to trace the signals internal to the chip made the mechanics of signal tracing peculiar to the individual design approaches. This meant that the maintenance personnel had to be trained to diagnose each peculiar system.

At that time the concept of developing tests for digital logic to detect logic faults was developed. [1] The stuck-at-one, stuck-at-zero fault concept was introduced and computer simulation methods were used to evaluate test sequence effectiveness.

Following the development of simulators, various methods for automatic test generation were developed. One of the first applications of such techniques was the development of tests for the A7-D/E aircraft avionics and its peculiar ground support equipment (PGSE).

For example, in this application [2] the logic schematics were encoded (Figure 1) as the source data for the logic simulation and test generation process. Figures 2 and 3 are a listing of the input code needed to generate tests for the unit. Figure 4 is the test sequence generated for the schematic and Figure 5 is the report of the test effectiveness for that test sequence. The test patterns are then converted to the ATE (automatic test equipment) source code and are ready for use on the ATE.

Such methods have been applied very successfully in the last decade on a number of aircraft programs. Boeing programs have made extensive use of various commercial test generation techniques as have virtually all major avionics manufacturers. [3] The military depots have, in like manner, used these techniques and have reported order of magnitude cost reductions over manual techniques. [4]

Since, the advent of LSI and VLSI (very large scale integration) technology and the application of microprocessors in systems, it has become increasingly difficult to develop the necessary simulation models needed to generate tests. The computer and model development costs can exceed the value of the equipment itself.

Because of the dynamic state of development in the semiconductor industry, models developed for simulation and test generation can become obsolete before the models can be developed and validated. In the past, once a "74/54" series part-library was developed, few changes were needed and the modeling costs could be written off against many users. In the current VLSI state of affairs, for custom designs, models

will probably be used only once, resulting in high set-up costs for test generation as well as expensive computer runs due to the large number of gates needed in the models.

Aside from the cost and modeling difficulties for the new microprocessor based designs, there is some question of the adequacy of the tests developed using the stuck fault concept.

Most test generation techniques are designed to develop static tests only. These simulators only approximate the time domain behavior of logic gates. Since 30 to 40 percent of the failure modes in VLSI are dynamic in nature, a test generation program reporting 100% of the "stuck-at" faults detected is probably only 70% effective for the total fault universe. This means that the remaining untested faults must be detected and isolated at the next higher assembly level. The usual symptom of such a problem is that the UUT (unit under test) will work in one system but not another. But when returned for test on the ATE, it always passes the test.

There are two other considerations for microprocessor systems that are unique. Such a system can be tested to exhaustion at the part level and subassembly level but yet fail in the system. Because microprocessor systems consist of hardware and firmware, testing the hardware and firmware independently does not mean that when the two are integrated the system will function as expected. Microprocessors exhibit failure modes, called pattern sensitivity, which are dependent upon the instruction sequence executed. Thus if two programmers were to write code for the same application, it is possible that one program could function normally and the other fail due to the peculiar sequence of instructions used by one programmer's code and not by the other.

For these reasons the built-in test concept, where the microprocessor exercises itself with a stored program, can only check out a fraction of the total failure modes.

The ideal approach is one where a complete checkout of the hardware and firmware, within the context of a dynamic test, can be performed. Further, the test generation technique should be capable of rapid modification as the system design matures, and be inexpensive to implement. Hopefully it would be a natural extension of the normal system synthesis and engineering process. The algorithmic test generation procedure is an approach which can meet these goals for a large class of systems.

ALGORITHMIC PATTERN GENERATION

Algorithmic pattern generators have found wide spread use in memory testing. Because of the highly ordered structures of memories, the efficiency of testing is greatly improved by using special hardware pattern generators instead of the classic truth tables. With the introduction of programmable pattern generators, millions of tests can be generated from a few input parameters. This also obviates the need to store the truth tables in expensive ATE.

As an example, to keep pace with memory development, a number of algorithms have been developed, i.e., "ping-pong", "butterfly", etc. Such techniques have reduced the cost of test equipment and made it possible to effectively test the most complex memories with a minimum of set-up costs. However, because of the differences in the design and manufacturing methods of memories, there is still no single completely accepted sequence of test algorithms available to completely test a particular semiconductor memory part.

The algorithms needed to perform a specific type of test for memories are well-known. For the type of systems under consideration, i.e., real-time microprocessor based control systems, no general pattern generation techniques have been developed to date. Most of the test procedures are based on exercising the instruction set of the microprocessor, or a simple input/output wrap-around test.

For the class of applications that are describable by transfer functions, it is possible to generate a set of algorithms that can create test patterns in a manner similar to that described for memory tests. Further, these patterns are dynamic in nature, and can be used to checkout the hardware and firmware to certify it's integrated adequately for a particular application.

The advantage of algorithmic approach is that faults in the system which have no effect on the intended use do not cause the unit to fail the test. For example, if a particular instruction of a microprocessor was faulty but it is not used, the fault is a 'don't care' fault. (If two programs were written for the same application and one programmer used the faulty instruction and the other did not and there were no coding errors, one would pass the test, the other would fail.) In like manner, if a memory cell were faulty and it was never used, the fault would be a 'don't care'. If the "stuck-at" test concept were used to develop tests, a useable unit could be rejected even though it could adequately perform the intended mission. Such rejections tend to increase the overall cost of systems.

The algorithmic test pattern generation concept tests the suitability of the total system for an application, including real time software and hardware performance. However, since the test procedure is application dependent, it does not assure the hardware is adequate for other uses. An advantage of the technique is that it is not difficult to synthesize the algorithms needed to create the desired test patterns for subsequent users.

ALGORITHM SYNTHESIS

Systems are frequently described in block diagram form with transfer functions as shown in Figure 6. Generally the system under test (SUT) and the "plant" or the controlled object are easily partitioned into independent and distinct transfer functions. For example, if the controller is digital, the transfer functions are usually represented in 'Z- Transform' form and the 'plant' in conventional LaPlace transforms. Generally, the systems analysis activities have developed numerical simulation equations for the system. But these are usually too complex or more detailed than needed for this application.

The first step of the process is to convert the SUT transfer functions into difference equations. These equations will be used to develop the outputs of the SUT, given the inputs. Since the Z-Transform parameters contain the sampling rate, the test rate is constrained to be that used in generating the Z-Transforms.

If the SUT transfer functions are LaPlace transforms, they must be converted to Z-Transforms. Generally, the step-invariant transform will provide an exact time-domain solution for most situations. The procedure for performing these mathematical operations are well-known. [5,6]

The difference equations developed at this step must be exact, because they form the basis for the algorithms that will generate test patterns for the SUT. The algorithms will probably generate test patterns more precise than those implemented in the microprocessor program itself, so the test procedure provides a quality control measure for the numerical methods used in the software. Such a quantitative measure is a valuable by-product of this testing approach.

Since the SUT generally has dynamic feedback signals from the plant being controlled, its transfer functions must likewise be translated into difference equations. These equations will generate the necessary feedback signals needed to dynamically test the SUT in the feedback loop.

However, these equations or models need not be as precise as those for the SUT, but need only be a reasonable approximation of the environment the SUT drives. Thus, in many situations the classic underdamped quadratic transfer function can be replaced by a single low-pass pole. Since stability analysis is not the goal, but rather the generation of feedback signals, such approximations greatly simplify the synthesis of the test generation algorithms without reducing the validity of the test procedure.

AN EXAMPLE

Figure 7 is a block diagram of an autopilot of an experimental missile, the T6. The T6 airframe is a 70 pound modular test vehicle designed to accept a variety of six inch diameter seekers. The T6 autopilot is typical of the type of real-time control system amenable to algorithmic test generation techniques.

For purposes of illustration the pitch channel will be used in the example, but the method can readily be extended to include all the channels and modes of operation. Figure 8 is a block diagram of the pitch channel of the autopilot. The first block is the pitch controller, the second block the actuator controller. Both of these functions are contained in the microprocessor software and are the objects of the test procedure. The other blocks are the missile fin actuator and the missile aerodynamics. The feedback to the autopilot is via a gyro which is assumed to have the transfer function of an integrator.

An algorithm for the first block is developed in equation (1) by expanding the Z-transform into a difference equation in the usual manner. The limiters are coded as shown in equation (2) and in like manner the actuator is expressed as shown in equation (3).

$$R(I)=[C(I)-S(I)]+(1-.9182)*[C(I-1)-S(I-1)]-0.9182*[C(I-2)-S(I-2)]-(.6434-.2302)*R(I-1)+(.2302)*(.6434)*R(I-2) \quad (1).$$

$$\text{IF } R(I) > 16/.18 \text{ THEN } R(I) = 16/.18 \quad (2).$$

$$\text{IF } R(I) < -16/.18 \text{ THEN } R(I) = -16/.18 \quad (3).$$

$$A(I) = R(I) - F(I) - .667*[R(I-1) - F(I-1)] \quad (4).$$

$$A(I) = .2184*A(I) \quad (5).$$

$$\text{IF } A(I) > 22.5/.18 \text{ THEN } A(I) = 22.5/.18 \quad (6).$$

$$\text{IF } A(I) < -22.5/.18 \text{ THEN } A(I) = -22.5/.18 \quad (7).$$

$$F(I) = .6*A(I) - .4*F(I) \quad (8).$$

$$C(I) = K*E*F(I) + 2*F*I - G*C(I-2) \quad (9).$$

Since the objective of the test is to checkout the autopilot hardware and software, a single pole low pass filter is used to represent the actuator in equation (8). A quadratic transfer function in equation (9) is derived from the Z-transform of the Laplace transfer function. An integrator is used to approximate the missile aerodynamics.

The scale factors are important (0.18 deg./LSB in the example) since these values are used to define the value of the least significant bit of the data generated by the tester. With the gains fixed as shown for the test procedure, the scale factors for the inputs and outputs are easily calculated.

To checkout the algorithms they are converted to a BASIC program and run on a computer. A listing of the program is provided in Figure 9, and the response to a step input is shown in Figure 10.

Once checked out using the BASIC program, the equations need to be modified slightly to be in the source language of the target tester. For this example the test system is Boeing's Sentry VIII test system (Figure 11). The source language is FACTOR, an ALGOL-type language. The conversion to FACTOR is a simple procedure.

To develop the detailed test program the microprocessor hardware interface must be defined. In this example an 8 bit data bus, one bit hold, and ready control are assumed. It should be understood that the detailed design of the particular SUT establishes the definitions. All inputs and outputs are multiplexed on the system data bus.

The first step is the definition of the input/output pins of the SUT. Also control pins are established along with the input output sequencer signals and the data bus control lines. The test rate is set up by tester commands in the manner peculiar to the individual tester. The tester driver is on when the "DA" command is active, off when the "DB" command is active. The tester is checking the output data when the "MA" is set and ignoring data when the "MB" is set. In the FACTOR code "DUT[DADB]=1" means the "DB" is set. "DUT[DADB]=0" means the "DA" is set. In like manner, "DUT[MAMB]=0" means the "MA" is set and "DUT[MAMB]=1" means "MB" is set. Other testers will use similar techniques, but it is important that these functions be performed in a dynamic manner (on-the-fly) otherwise the test data will not be valid.

The FACTOR source code is shown in Figure 11, and a compiled listing in Figure 12. The generated test code is shown in Figure 13. It should be noted that the actual code is significantly longer and only a small portion of the actual code is printed.

GENERALIZATIONS & CONCLUSIONS

The algorithms developed in the example were selected to illustrate the techniques of "algorithmic pattern generation" and only illustrate one of many possible approaches to modeling systems for test generation purposes. The recursive difference equation, based on the Z-Transform calculus, has many advantages in that the theory is highly developed, and is a very natural way to develop a dynamic test.

Those versed in simulation art will find many ways to construct algorithms that do not require transfer functions or the linear system assumptions.

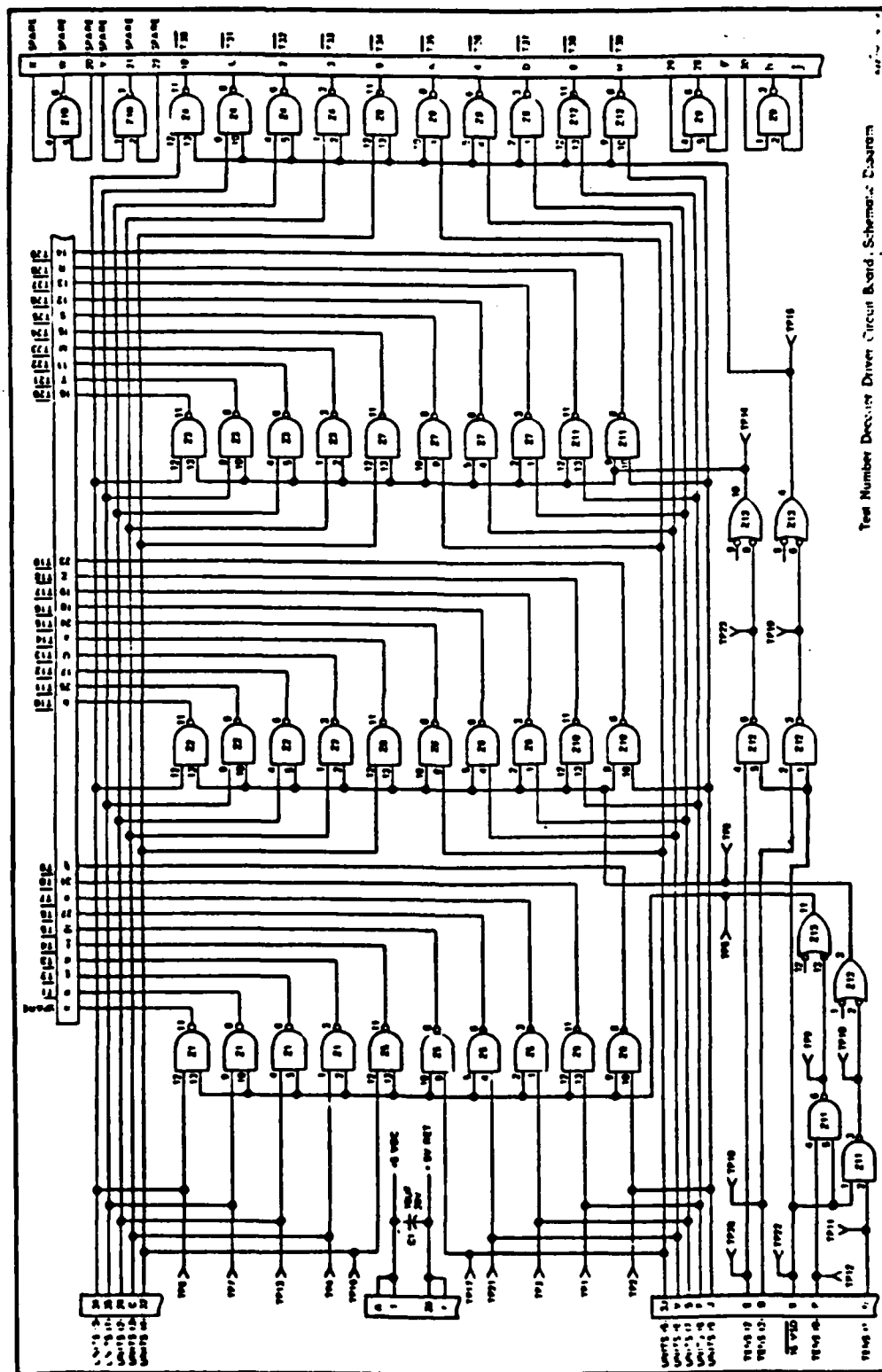
The nature of the algorithms is constrained to a degree by the target tester. The system used in the example, the Sentry VIII manufactured by Fairchild, is particularly easy to use to create test algorithms. The utility program "LMLRN" handles many of the tester control functions in a very simple manner. Other testers may require an additional processing step to perform the test control generation function.

Although not mentioned in the example, it is feasible to expand the input/output definitions (pin lists) and 'learn' the test data at the subassembly levels to form the basis for developing production and field support tests. The tester control functions for such an application would be significantly more complex, but with careful design of the subsystem interfaces, the process can be made very productive.

The approach outlined is compatible with the normal systems engineering approach. Simulation models used to support systems analysis activities are directly applicable to the test generation process. However, further research and analysis is needed to define the adequacy of test procedures developed using functional approaches instead of the classic "stuck-at" methods. The acceptance of equipment with possible manufacturing defects, even though the defects do not impact the intended function, presents certain cultural problems to a quality assurance operation, i.e. "Zero-Defects".

REFERENCES

- (1) Seshu, S. and Freeman, D.N., "The Diagnosis of Asynchronous Sequential Switching Systems", IRE Trans. Electronic Computers, vol. EC-11, pp. 459-465, August 1962.
- (2) TO 51T1-15-8-29, "Program number 47264"
- (3) "SELECTION GUIDE FOR DIGITAL TEST PROGRAM GENERATION SYSTEMS", Navmatinst 3960.9A, 27 MARCH 1979.
- (4) Pedersen, G. "Reducing ATE Software Development Costs", AUTOTESTCON, Sept. 1979, Minneapolis, Minn.
- (5) Breeding, D.R. "Digital Simulation of a Seismic System", Proceedings of the 26'th International Instrumentation Symposium, Seattle, Wa., Advances in Test Measurement, Vol 17, pp. 297-299.
- (6) Stearns, S.D. 1975, "Exact Digital Models of Continuous Linear Systems", SAND75-8012.



Test Number Decoder/Driver Circuit Board Schematic Diagram

Figure 1. Test Number Decoder/Driver Circuit Board Schematic Diagram—Drawing Number 275280

FILE	LOC	QWCODE	ADDR	TYPE	SOURCE	STATEMENT
	754	754	0000	00	0000	
	756	756	0000	00	0000	
	758	758	0000	00	0000	
	760	760	0000	00	0000	
	762	762	0000	00	0000	
	764	764	0000	00	0000	
	766	766	0000	00	0000	
	768	768	0000	00	0000	
	770	770	0000	00	0000	
	772	772	0000	00	0000	
	774	774	0000	00	0000	
	776	776	0000	00	0000	
	778	778	0000	00	0000	
	780	780	0000	00	0000	
	782	782	0000	00	0000	
	784	784	0000	00	0000	
	786	786	0000	00	0000	
	788	788	0000	00	0000	
	790	790	0000	00	0000	
	792	792	0000	00	0000	
	794	794	0000	00	0000	
	796	796	0000	00	0000	
	798	798	0000	00	0000	
	800	800	0000	00	0000	
	802	802	0000	00	0000	
	804	804	0000	00	0000	
	806	806	0000	00	0000	
	808	808	0000	00	0000	
	810	810	0000	00	0000	
	812	812	0000	00	0000	
	814	814	0000	00	0000	
	816	816	0000	00	0000	
	818	818	0000	00	0000	
	820	820	0000	00	0000	
	822	822	0000	00	0000	
	824	824	0000	00	0000	
	826	826	0000	00	0000	
	828	828	0000	00	0000	
	830	830	0000	00	0000	
	832	832	0000	00	0000	
	834	834	0000	00	0000	
	836	836	0000	00	0000	
	838	838	0000	00	0000	
	840	840	0000	00	0000	
	842	842	0000	00	0000	
	844	844	0000	00	0000	
	846	846	0000	00	0000	
	848	848	0000	00	0000	
	850	850	0000	00	0000	
	852	852	0000	00	0000	
	854	854	0000	00	0000	
	856	856	0000	00	0000	
	858	858	0000	00	0000	
	860	860	0000	00	0000	
	862	862	0000	00	0000	
	864	864	0000	00	0000	
	866	866	0000	00	0000	
	868	868	0000	00	0000	
	870	870	0000	00	0000	
	872	872	0000	00	0000	
	874	874	0000	00	0000	
	876	876	0000	00	0000	
	878	878	0000	00	0000	
	880	880	0000	00	0000	
	882	882	0000	00	0000	
	884	884	0000	00	0000	
	886	886	0000	00	0000	
	888	888	0000	00	0000	
	890	890	0000	00	0000	
	892	892	0000	00	0000	
	894	894	0000	00	0000	
	896	896	0000	00	0000	
	898	898	0000	00	0000	
	900	900	0000	00	0000	
	902	902	0000	00	0000	
	904	904	0000	00	0000	
	906	906	0000	00	0000	
	908	908	0000	00	0000	
	910	910	0000	00	0000	
	912	912	0000	00	0000	
	914	914	0000	00	0000	
	916	916	0000	00	0000	
	918	918	0000	00	0000	
	920	920	0000	00	0000	
	922	922	0000	00	0000	
	924	924	0000	00	0000	
	926	926	0000	00	0000	
	928	928	0000	00	0000	
	930	930	0000	00	0000	
	932	932	0000	00	0000	
	934	934	0000	00	0000	
	936	936	0000	00	0000	
	938	938	0000	00	0000	
	940	940	0000	00	0000	
	942	942	0000	00	0000	
	944	944	0000	00	0000	
	946	946	0000	00	0000	
	948	948	0000	00	0000	
	950	950	0000	00	0000	
	952	952	0000	00	0000	
	954	954	0000	00	0000	
	956	956	0000	00	0000	
	958	958	0000	00	0000	
	960	960	0000	00	0000	
	962	962	0000	00	0000	
	964	964	0000	00	0000	
	966	966	0000	00	0000	
	968	968	0000	00	0000	
	970	970	0000	00	0000	
	972	972	0000	00	0000	
	974	974	0000	00	0000	
	976	976	0000	00	0000	
	978	978	0000	00	0000	
	980	980	0000	00	0000	
	982	982	0000	00	0000	
	984	984	0000	00	0000	
	986	986	0000	00	0000	
	988	988	0000	00	0000	
	990	990	0000	00	0000	
	992	992	0000	00	0000	
	994	994	0000	00	0000	
	996	996	0000	00	0000	
	998	998	0000	00	0000	
	1000	1000	0000	00	0000	

Figure 3. Network (Concluded)

E

INPUTS

DATE: 06/10/77									
TIME: 16:36:50									
PALM MILD MACROLOGIC - TEST VERIFICATION/GENERATION PROGRAM									
RESULT SUMMARY									
NEWMELL 2244									
PAGE 33									
INDIVIDUAL DEFECTS									
TYPE	INCLUDING S-SIGNALS				EXCLUDING S-SIGNALS				
	TOTAL NUMBER CONSIDERED	NUMBER DETECTED	PERCENT DETECTED	UNDEFINED OUTPUT	TOTAL NUMBER CONSIDERED	NUMBER DETECTED	PERCENT DETECTED	UNDEFINED OUTPUT	
STUCK OUTPUTS	238	238	100.00	0	238	238	100.00	0	
STUCK INPUTS	334	334	100.00	0	334	334	100.00	0	
SHORTS	0	0	0.00	0	0	0	0.00	0	
OPENS	0	0	0.00	0	0	0	0.00	0	
TOTAL	572	572	100.00	0	572	572	100.00	0	
EQUIVALENCE CLASSES									
TYPE	INCLUDING S-SIGNALS				EXCLUDING S-SIGNALS				
	TOTAL NUMBER CONSIDERED	NUMBER DETECTED	PERCENT DETECTED	UNDEFINED OUTPUT	TOTAL NUMBER CONSIDERED	NUMBER DETECTED	PERCENT DETECTED	UNDEFINED OUTPUT	
CLASSES INCLUDING AT LEAST ONE STUCK OUTPUT DEFECT	190	190	100.00	0	190	190	100.00	0	
CLASSES INCLUDING AT LEAST ONE STUCK INPUT DEFECT	242	242	100.00	0	242	242	100.00	0	
CLASSES INCLUDING AT LEAST ONE SHORT DEFECT	0	0	0.00	0	0	0	0.00	0	
CLASSES INCLUDING AT LEAST ONE OPEN DEFECT	0	0	0.00	0	0	0	0.00	0	
ALL EQUIVALENCE CLASSES	246	246	100.00	0	246	246	100.00	0	

Figure 5. Test Program Scope

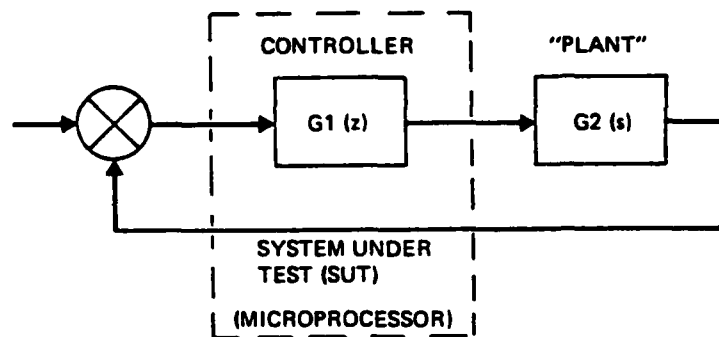


Figure 6. Block Diagram—Typical System

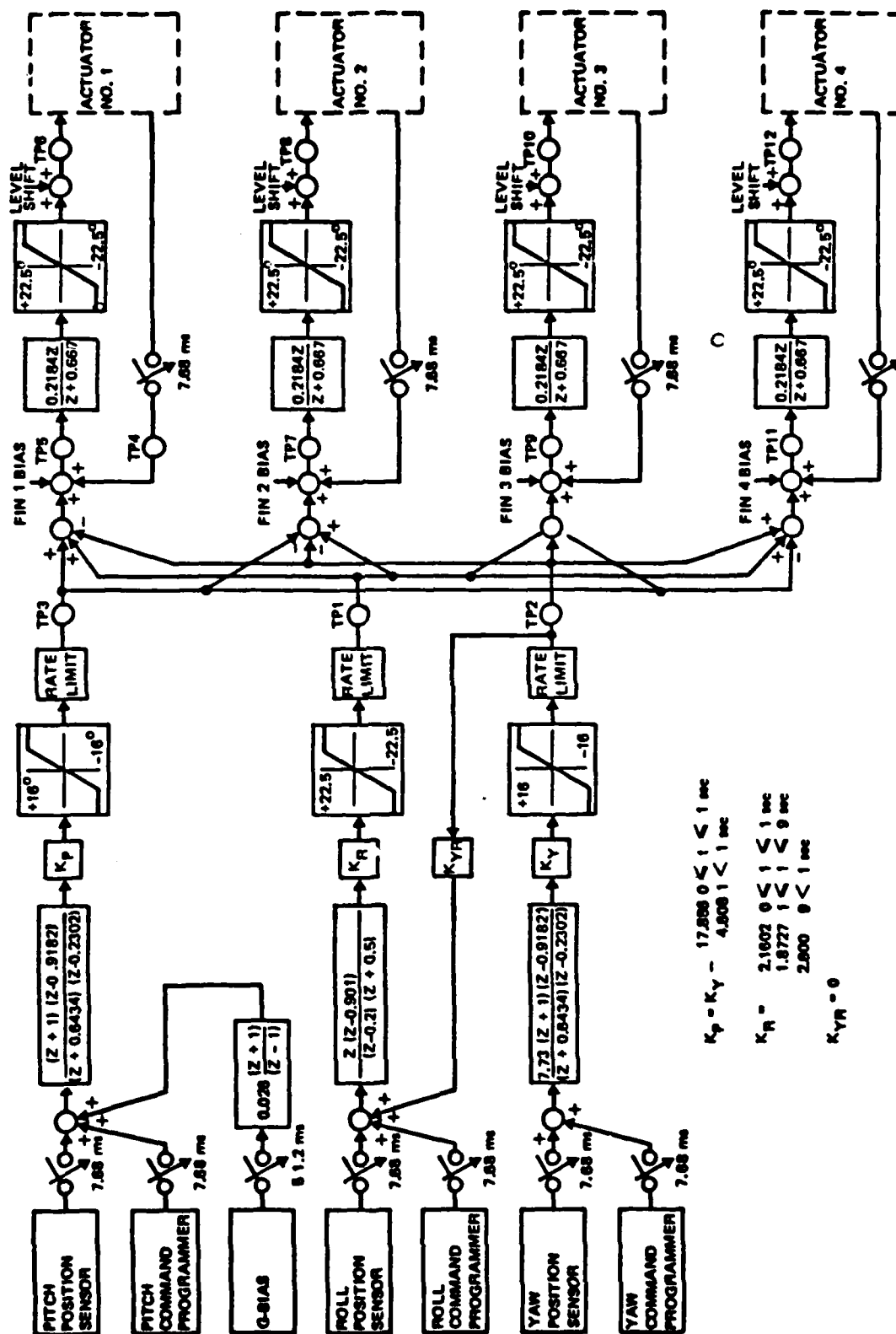


Figure 7. Autopilot Functional Diagram

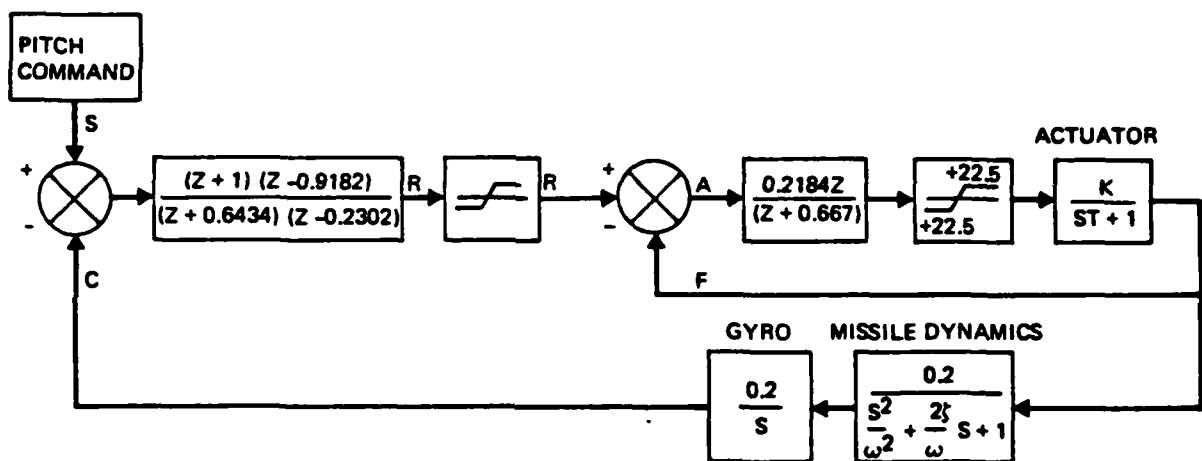


Figure 8. Block Diagram of Pitch Channel

FIGURE 9 "BASIC" SOURCE CODE FOR TEST ALGORITHM CHECKOUT

```

1 REM *****
2 REM   DEFINE SIMULATION VARIABLES
3 REM
4 REM   R=AUTOPILOT CONTROLLER OUTPUT
5 REM   C=MISSILE FEEDBACK
6 REM   A=ACTUATOR CONTROL SIGNAL
7 REM   F=FIN FEEDBACK SIGNAL
8 REM   S=COMMAND INPUT
9 REM *****
10 DIM R(3),C(3),A(3),F(3),S(3),X(3)
11 REM *****
12 REM   INITIALIZE ALL VARIABLES
13 REM *****
14 MAT R=ZER
15 MAT C=ZER
16 MAT A=ZER
17 MAT F=ZER
18 MAT S=ZER
19 MAT X=ZER
20 REM *****
21 REM   SIMULATION OF CONTROLLER USING DIFFERENCE EQUATIONS
22 REM *****
23 REM
24 I=3
25 FOR J=1 TO 100
26 IF J=3 THEN S(3)=10
27 R(I)=C(I)+.0818*C(I-1)-.9182*C(I-2)-.4132*R(I-1)+.1481*R(I-2)
28 REM *****
29 REM   GAIN AND LIMIT CALCULATIONS FOR CONTROLLER
30 REM *****
31 IF R(I)>16/.18 THEN R(I)=16/.18
32 IF R(I)<-16/.18 THEN R(I)=-16/.18
33 REM *****
34 REM   ACTUATOR DIFFERENCE EQUATIONS
35 REM *****
36 A(I)=R(I)-F(I)-.667*(R(I-1)-F(I-1))
37 A(I)=.2184*A(I)
38 REM *****
39 REM   ACTUATOR GAIN AND LIMIT CALCULATIONS
40 REM *****
41 IF A(I)>22.5/.18 THEN A(I)=22.5/.18
42 IF A(I)<-22.5/.18 THEN A(I)=-22.5/.18
43 REM *****
44 REM   FIN SERVOCALCULATIONS
45 REM *****
46
47
48

```

```

480 REM
500 F(I)=.6*A(I)-.4*F(I)
510 REM
520 REM *****
530 REM MISSILE FLIGHT CALCULATIONS
531 REM
532 REM MISSILE NATURAL FREQ.=120 RADIANS/SEC.
533 REM
534 REM MISSILE DAMPING FACTOR =.7
535 REM
536 REM GAIN CONSTANT = .2
540 REM *****
550 REM
560 K=1.24882
570 E=.313568
580 F=.458459
590 G=.30851
600 C(I)=.2*K*E*F(I)+2*F*C(I-1)-G*C(I-2)
605 X(I)=.2*C(I)+X(I)
606 C(I)=S(I)-X(I)
610 REM
620 REM *****
630 REM TUMBLE VARIABLES
640 REM *****
650 REM
655 PRINT USING 741;J,S(3),X(3),R(3),A(3),F(3),C(3)
660 R(1)=R(2)
670 R(2)=R(3)
680 C(1)=C(2)
690 C(2)=C(3)
700 A(1)=A(2)
710 A(2)=A(3)
720 F(1)=F(2)
730 F(2)=F(3)
741 IMAGE 7(XDDD.DX)
800 NEXT J
9000 END

```

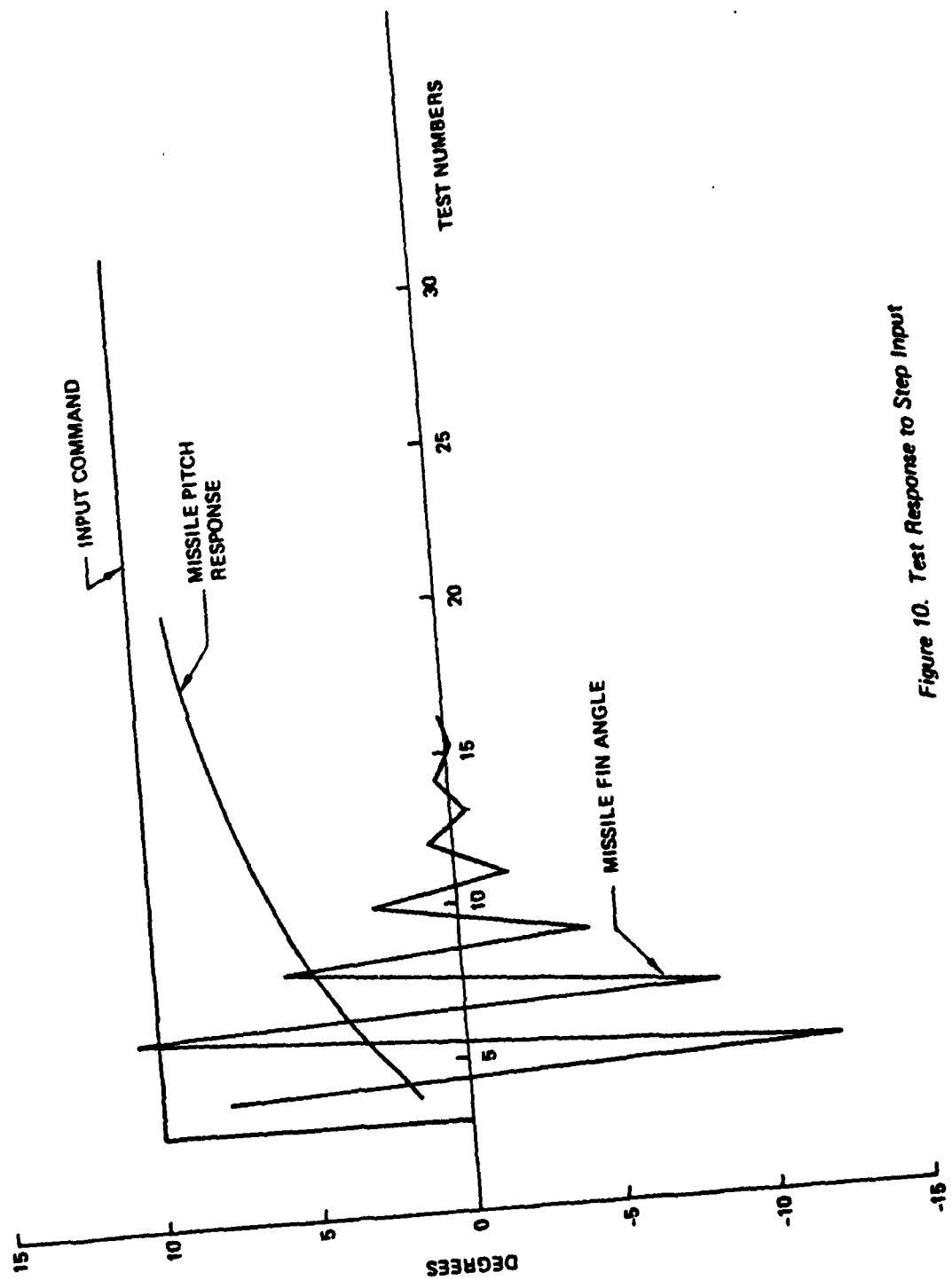


Figure 10. Test Response to Step Input

FIGURE 11 ALGORITHMIC TEST PATTERNS FOR SENTRY VIII TESTER

```

SET PAGE 4095;

REM *****
* SET UP PIN DEFINITIONS FOR ALGORITHMIC *
* TEST PATTERN GENERATION *
*****;

DCL PDATA<8>/10,11,12,13,14,15,16,17/;
DCL PHLD<1>/5/;
DCL PRDY<1>/3/;

DCL DUT<5>;

DATA=1;
HLD=2;
RDY=3;
DADB=4;
MAMB=5;

EXEC LMMOD(0); REM INITIALIZE TESTER MEMORY;

EXEC LMLRN(1,PDATA,PHLD,PRDY,1,2);

REM SET UP PIN DEFINITIONS IN TESTER;

REM *****
* DEFINE SIMULATION VARIABLES *
*****;

DCL R<3>;
DCL C<3>;
DCL A<3>;
DCL F<3>;
DCL S<3>;
DCL X<3>;

REM *****
* SIMULATION OF CONTROLLER USING DIFF. EQU. *
*****;

I=3;
DUT<HLD>=0; REM START PROCESSOR;
DUT<RDY>=1;
DUT<DADB>=0;
DUT<MAMB>=1;

EXEC LMLRN(4,M,DUT); REM GENERATE A TEST;

```

```

M=M+1;                                REM TEST NUMBER COUNTER;

FOR J=1 THRU 100 DO BEGIN
IF J EQ 3 THEN S<I>=10;

R<I>=C<I>+.0818*C<I-1>-.9182*C<I-2>-.4132*R<I-1>
+.1481*R<I-2>;

REM*****
* GAIN AND LIMIT CALCULATIONS *
*****;

IF R<I> GT 16/.18 THEN R<I>=16/.18;
IF R<I> LT -16/.18 THEN R<I>=-16/.19;

REM*****
* ACTUATOR DIFFERENCE EQUATIONS *
*****;

A<I>=R<I>-F<I>-.667*(R<I-1>-F<I-1>);
A<I>=.2184*A<I>;

REM*****
* ACTUATOR GAIN AND LIMIT CALCULATIONS *
*****;

IF A<I> GT 22.5/.18 THEN A<I>=22.5/.18;
IF A<I> LT -22.5/.18 THEN A<I>=-22.5/.18;

REM*****
* FIN SERVO CALCULATIONS *
*****;

F<I>=.6*A<I>-.4*F<I>;

REM*****
* MISSILE FLIGHT CALCULATIONS *
* * * * *
* NATURAL FREQUENCY= 120 RADIAN/SEC. *
* * * * *
* DAMPING FACTOR=.7 *
* * * * *
* GAIN=.2 *
*****;

L=1.24882;
E=0.313568;
F=0.458459;
G=0.30851;

C<I>=.2*K*E*F<I>+2*L*C<I-1>-G*C<I-2>;
X<I>=.2*C<I>+X<I>;
C<I>=S<I>-X<I>;

```

```

REM*****
*   TUMBLE VARIABLES   *
*****;

```

```

R<1>=R<2>;
R<2>=R<3>;
C<1>=C<2>;
C<2>=C<3>;
A<1>=A<2>;
A<2>=A<3>;
F<1>=F<2>;
F<2>=F<3>;

```

```

REM*****
*           INPUT/OUTPUT SEQUENCES           *
*                                           *
*   1) INPUT COMMAND (TESTER DRIVES BUS)   *
*   2) FIN  COMMAND (SUT  DRIVES BUS)     *
*   3) FIN FEEDBACK (TESTER DRIVES BUS)    *
*   4) GYRO FEEDBACK (TESTER DRIVES BUS)    *
*****;

```

```

REM INPUT 1'ST INPUT PATTERN;

```

```

DUT<DATA>=S<I>/.18 AND 377B; REM RESCALES AND COVERTS
DUT<DADB>=0;
DUT<MAMB>=1; REM TESTER INPUTTING DATA ON THE BUS
                TO AN INTEGER NUMBER;

```

```

EXEC LMLRN(4,M,DUT); REM GENERATES A TEST;
M=M+1;

```

```

REM OUTPUT ACTUATOR COMMAND;

```

```

DUT<DATA>=A<I>/.18 AND 377B;
DUT<DADB>=1;
DUT<MAMB>=0; REM TESTER LOOKING AT DATA ON BUS;
EXEC LMLRN(4,M,DUT);
M=M+1;

```

```

REM INPUT FEEDBACK FROM FIN SERVO;

```

```

DUT<DATA>=F<I>/.18 AND 377B;
DUT<DADB>=0; REM TESTER DRIVES THE BUS;
DUT<MAMB>=1;
EXEC LMLRN(4,M,DUT);
M=M+1;

```

```

REM INPUT GYRO FEEDBACK;

```

```

DUT<DATA>=C<I>/.18 AND 377B;
EXEC LMLRN(4,M,DUT);
M=M+1;

```

```

END; REM END OF FOR LOOP;

```

END;

REM END OF TEST PROGRAM;

```

**** FAIRCHILD FACTOR COMPILER REL: 3.1 ****
** SOURCE FILE: *ED DATE: **
** DATA FILE: TIME: 00:35 **
** SYSTEM CONF: 88 MV1 PMU4 COND: 00000000 **

1 SET PAGE 4095;
2
2 REM *****
2 * SET UP PIN DEFINITIONS FOR ALGORITHMIC *
2 * TEST PATTERN GENERATION *
2 *****;
2
2 DCL PDATA[8]/10, 11, 12, 13, 14, 15, 16, 17//;
3 DCL PHLD[1]/5//;
4 DCL PRDY[1]/3//;
5
5 DCL DUT[5];
6
6 DATA=1;
7 HLD=2;
8 RDY=3;
9 DADB=4;
10 MAMB=5;
11
11 EXEC LMMOD(0); REM INITIALIZE TESTER MEMORY;
12
12 EXEC LMLRN(1, PDATA, PHLD, PRDY, 1, 2);
13
13 REM SET UP PIN DEFINITIONS IN TESTER;
13
13 REM *****
13 * DEFINE SIMULATION VARIABLES *
13 *****;
13
13 DCL R[3];
14 DCL C[3];
15 DCL A[3];
16 DCL F[3];
17 DCL S[3];
18 DCL X[3];
19
19 REM *****
19 * SIMULATION OF CONTROLLER USING DIFF. EQU. *
19 *****;
19
19 I=3;
20 DUT[HLD]=0; REM START PROCESSOR;
21 DUT[RDY]=1;
22 DUT[DADB]=0;
23 DUT[MAMB]=1;
24
24 EXEC LMLRN(4, M, DUT); REM GENERATE A TEST;
25
25 M=M+1; REM TEST NUMBER COUNTER;
26
26 FOR J=1 THRU 25 DO BEGIN
27 IF J EQ 3 THEN S[I]=10;
29
29 R[I]=C[I]+.0818*C[I-1]-.9182*C[I-2]-.4132*R[I-1]
30 +.1481*R[I-2];
30
30 REM *****

```

Figure 12. Compile Source Code

```

30          * GAIN AND LIMIT CALCULATIONS          *
30
30          *****
30          IF R[I] GT 16/.18 THEN R[I]=16/.18;
32          IF R[I] LT -16/.18 THEN R[I]=-16/.18;
34
34          REM*****
34          * ACTUATOR DIFFERENCE EQUATIONS          *
34          *****
34          A[I]=R[I]-F[I]-.667*(R[I-1]-F[I-1]);
35          A[I]=.2184*A[I];
36
36          C          REM*****
36          * ACTUATOR GAIN AND LIMIT CALCULATIONS          *
36          *****
36
36          IF A[I] GT 22.5/.18 THEN A[I]=22.5/.18;
38          IF A[I] LT -22.5/.18 THEN A[I]=-22.5/.18;
40
40          REM*****
40          * FIN SERVO CALCULATIONS          *
40          *****
40
40          F[I]=.6*A[I]-.4*F[I];
41
41          REM*****
41          * MISSILE FLIGHT CALCULATIONS          *
41          *
41          * NATURAL FREQUENCY= 120 RAD/SEC.          *
41          *
41          * DAMPING FACTOR=.7          *
41          *
41          * GAIN=.2          *
41          *****
41
41          K=1.24882;
42          E=0.313568;
43          L=0.458459;
44          O=0.30851;
45
45          C[I]=.2*K*E*F[I]+2*L*C[I-1]-O*C[I-2];
46          X[I]=.2*C[I]+X[I];
47          C[I]=S[I]-X[I];
48
48          REM*****
48          * TUMBLE VARIABLES          *
48          *****
48
48          R[1]=R[2];
49          R[2]=R[3];
50          C[1]=C[2];
51          C[2]=C[3];
52          A[1]=A[2];
53          A[2]=A[3];
54          F[1]=F[2];
55          F[2]=F[3];
56
56          WRITE(POD)R[3],C[3],A[3],F[3],A[3]/.18 AND 377B,(A[3]/.18) AND 3
77B;
57
57          REM*****
57          * INPUT/OUTPUT SEQUENCES          *

```

Figure 12. Compile Source Code (Continued)

0 SET F
 1 SET F
 2 SET F
 3 SET F
 4 SET F
 5 SET F
 6 SET F
 7 SET F
 8 SET F
 9 SET F
 10 SET F
 11 SET F
 12 SET F
 13 SET F
 14 SET F
 15 SET F
 16 SET F
 17 SET F
 18 SET F
 19 SET F
 20 SET F
 21 SET F
 22 SET F
 23 SET F
 24 SET F
 25 SET F
 26 SET F
 27 SET F
 28 SET F
 29 SET F
 30 SET F
 31 SET F
 32 SET F
 33 SET F
 34 SET F
 35 SET F
 36 SET F
 37 SET F
 38 SET F
 39 SET F
 40 SET F
 41 SET F
 42 SET F
 43 SET F
 44 SET F
 45 SET F
 46 SET F
 47 SET F
 48 SET F
 49 SET F
 50 SET F
 51 SET F
 52 SET F
 53 SET F
 54 SET F
 55 SET F
 56 SET F
 57 SET F
 58 SET F
 59 SET F
 60 SET F
 61 SET F

DA MB	0010000000	0000000000	0000000000
DA MB	0010000000	0000000000	0000000000
DB MA	0010000000	0000000000	0000000000
DA MB	0010000000	0000000000	0000000000
DA MB	0010000000	0000000000	0000000000
DA MB	0010000000	0000000000	0000000000
DB MA	0010000000	0000000000	0000000000
DA MB	0010000000	0000000000	0000000000
DA MB	0010000000	0000000000	0000000000
DA MB	0010000000	0011100000	0000000000
DB MA	0010000000	0000000000	0000000000
DA MB	0010000000	0000000000	0000000000
DA MB	0010000000	0011100000	0000000000
DB MA	0010000001	0110000000	0000000000
DA MB	0010000000	0010000000	0000000000
DA MB	0010000001	0110100000	0000000000
DB MA	0010000000	0011100000	0000000000
DA MB	0010000001	0001111000	0000000000
DA MB	0010000000	0101111000	0000000000
DA MB	0010000001	0010100000	0000000000
DA MB	0010000000	0011100000	0000000000
DB MA	0010000000	1010000000	0000000000
DA MB	0010000001	1010000000	0000000000
DA MB	0010000000	0100100000	0000000000
DA MB	0010000000	0011100000	0000000000
DB MA	0010000001	0011111000	0000000000
DA MB	0010000000	0011111000	0000000000
DA MB	0010000000	0000100000	0000000000
DA MB	0010000000	0011100000	0000000000
DB MA	0010000000	0100000000	0000000000
DA MB	0010000000	1100000000	0000000000
DA MB	0010000000	0111000000	0000000000
DA MB	0010000000	0011100000	0000000000
DB MA	0010000001	0111111000	0000000000
DA MB	0010000000	0111111000	0000000000
DA MB	0010000001	0011000000	0000000000
DA MB	0010000000	0011100000	0000000000
DB MA	0010000000	1000000000	0000000000
DA MB	0010000001	1000000000	0000000000
DA MB	0010000000	1101000000	0000000000
DA MB	0010000000	0011100000	0000000000
DB MA	0010000001	1111111000	0000000000
DA MB	0010000000	1111111000	0000000000
DA MB	0010000000	0101000000	0000000000
DA MB	0010000000	0011100000	0000000000
DB MA	0010000001	0000000000	0000000000
DA MB	0010000001	0000000000	0000000000
DA MB	0010000001	0001000000	0000000000
DA MB	0010000000	0011100000	0000000000
DB MA	0010000001	1111111000	0000000000
DA MB	0010000001	1111111000	0000000000
DA MB	0010000001	1110000000	0000000000
DA MB	0010000000	0011100000	0000000000
DB MA	0010000000	0000000000	0000000000
DA MB	0010000001	0000000000	0000000000
DA MB	0010000000	1110000000	0000000000
DA MB	0010000000	0011100000	0000000000
DB MA	0010000000	0000000000	0000000000
DA MB	0010000000	0000000000	0000000000
DA MB	0010000000	0110000000	0000000000
DA MB	0010000000	0011100000	0000000000

Figure 13. Resulting Test Sequence