1988

A general IC test program structure /

Ernst J. Wahl
Lehigh University

A General IC Test Program Structure

by
Ernst J. Wahl

A Thesis
Presented to the Graduate Committee
of Lehigh University
in Candidacy for the Degree of

Master of Science
in
Electrical Engineering

Lehigh University
1988

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

May 20, 1988
Date

D. Richard Decker
Professor in Charge

L. J. Vanevenn
Chairman of Department

ACKNOWLEDGMENTS

Thanks to Mark Barber for providing constructive criticism and allowing me the time to develop GENTEST. I thank Roy Odhner and Carey Garrenton for their advice and help in overcoming software obstacles and testing problems, and their contributions of utility subroutines CHANNL and SHOP. I also thank Jimmy Tan, Bob Fisher, Roger Jeffery, Rich Lewis, Jim Scorzelli, and Ray Yock for providing useful feedback and suggestions.

Ernst J. Wahl

CONTENTS

1. INTRODUCTION 2

2. TEST GENERATION STRATEGY 4

2.1 What are the Obstacles? 4

2.2 Approaching the Problem 6

3. GENERAL TEST PROGRAM OBJECTIVES 6

4. PROGRAMMING GUIDELINES 7

5. GENTEST STRUCTURE 10

5.1 Test Language Dependency 10

5.2 Test Routine Structure 10

5.3 GENTEST Modules 11

5.4 File Naming Conventions 13

5.5 File Types 14

5.6 File Contents 14

6. RESULT 18

6.1 GENTEST Attributes 18

6.1.1 Test Program Generation 19

6.1.2 Debugging and Fine Tuning 21

6.1.3 Operation and Maintenance 21

6.1.4 Data Collection 22

6.2 GENTEST Applied 22

6.3 The ATE Vendor and GENTEST 23

6.4 Summary 23

REFERENCES 25

7. APPENDIX: Glossary of Terms 27

8. VITA 52

LIST OF FIGURES

Figure 1. Current Test Program Generation Approach 32

Figure 2. Proposed Test Program Generation Approach 33

Figure 3. A Hierarchical Representation of Commonly Performed Tests 35

Figure 4. Interrelation of GENTEST Modules 35

Figure 5. Interrelation of GENTEST Files 37

Figure 6. File Contents: Insert Files 37

Figure 7. File Contents: MAIN Program 38

Figure 8. File Contents: Subroutines 39

Figure 9. Flow Chart: MAIN Program 41

Figure 10. Flow Chart: Typical Test Subroutine 41

Figure 11. Subroutine GMENU: Characterization & Program Development Mode 42

Figure 12. Subroutine GMENU: Production Mode 42

Figure 13. Subroutine ACC: Timing Measurement Output, Debug Mode 43

Figure 14. Subroutine ACC: Timing Measurement Engineering Output 45

Figure 15. Subroutine DCC: DC Measurement Output 46

Figure 16. Subroutine DIODE: Diode Test Output 48

Figure 17. Subroutine FNC: Functional Test Output 48

Figure 18. Subroutines FQBC or FQLC: Frequency Test Output 49

Figure 19. Subroutine FQLC: Linear Search Frequency Test Output 50

Figure 20. Compact Characterization Test Output 51

ABSTRACT

One of the goals of IC design and manufacturing is to develop an IC test generation strategy that will produce consistently high quality characterization and production type tests for a wide range of devices in a cost effective manner. Previous publications on testing have emphasized the development of automatic TPGs (*Test Program Generators*). TPGs have successfully reduced the cost of developing and maintaining many test programs, but tend to be inflexible in terms of meeting diverse testing requirements.

This thesis suggests that the structure of the test program, generated by a TPG, impacts the effectiveness of the test generation strategy. A test program structure that separates general and device-dependent source code, is proposed. The general code is reusable whereas the device-dependent code is replaced for each new device. An implementation of this structure has been written in ATL (*Advanced Test Language*) to run on an Advantest® T3340 test system.

This method permits simultaneous development of a characterization and a production test program by allowing one database to drive both. The structure is modular to encourage a one-to-one correspondence between the generated test program and the CRT image presented to the user by the TPG. This helps to reduce development time. Default program flow and test definitions are readily altered to fit special requirements. Debugging hooks are provided where test system utilities fall short. Depending on a hardware switch setting, the test program operates in either production or characterization mode.

1. INTRODUCTION

This paper addresses problems associated with testing digital ICs. Since there are numerous methods of and reasons for testing, some clarification may be in order.

Testing, as presented here, follows a device from development through production. The device may be an SSI or VLSI circuit. The tool of choice is a test system comprised of hardware and software components commonly referred to as ATE (*automatic test equipment*). The hardware consists of a computer (CRT, CPU, memory, disk, tape), a test head which contains electronics to stimulate the DUT (*device under test*) and relay the DUT's response, and a cast of supporting hardware. The software consists of an operating system, a test language, and supporting utilities.

The tests performed on the DUT fall into two categories: functional and parametric. The parametric tests may further be subdivided into AC and DC tests. The functional tests assure that the DUT conforms to truth tables called vector loads. Vector load source files contain symbols representing logic states which are used to stimulate the DUT and anticipate its response. AC tests consist of timing measurements such as set-up and hold times and propagation delay. DC tests consist of various voltage and current measurements. Other tests may be derived from the major types. For example, the repeated application of a functional test to determine the maximum clock rate at which the DUT will operate, is referred to as a frequency test.

As a device progresses from development to production the emphasis changes from device characterization to device classification. The associated test implementations are drastically different since the former requires time consuming data gathering whereas the latter requires speedy execution.

The purpose of these tests is to ensure the functionality of the DUT in the environment for which it was designed by gathering data in the environment of the ATE. The two

environments usually have different electric properties. There is ample room for the application of transmission line theory to clarify problems associated with the DUT/ATE interface, control system theory to address problems associated with ATE power supply response to DUT demands, and other forms of ingenuity to deal with problems arising from the fact that at any point in time, state of the art devices are tested with older technology.

Historically, test engineers would manually write a device characterization test program, if necessary, and then a production test program. This turned out to be too time consuming and therefore expensive, so attempts were made to define standard test languages, standard test modules, and automatic TPGs (*Test Program Generators*). These efforts have produced such test program generation tools as ATLAS[1] (Test Language), ICTEST[2] (Test Language), and ALPHA II[3] (Test Program Generator). At AT&T Bell Laboratories, similar efforts have produced TIF[4] (Tester Independent Format), Sentry Test Modules[5] , TPG[6] [7] (Test Program Generator), TPG2[8] [9], and ATGEN[10] (Automatic Test Program Generator), of which TPG2 and ATGEN are still in use. TPG2 interprets CAD tool output and standard libraries to generate test programs suitable for production. ATGEN allows test engineers to enter data via a form editor and generates test programs suitable for device characterization.

The TPGs, TPG2 and ATGEN, successfully reduced the expense of developing and maintaining IC test programs for a large number of devices, but problems remained. Test engineers had to learn two very different systems in order to generate both characterization and production programs. It was difficult to determine whether the two programs allocated system resources in the same way. Test program adaptations to comply with device requirements for which the TPGs were unprepared, were awkward to implement. For these reasons the author developed a *general test program* named GENTEST. This program consists of a variety of original and borrowed code and ideas, adapted to fit into a modular structure. The main feature of GENTEST is the sharing of databases between test routines

which perform diverse functions such as characterization and production testing. GENTEST programs are currently generated by manually providing device-dependent information in the appropriate files and then translating the corresponding routines. The author's suggestion is that TPGs could benefit by generating a test program utilizing this type of structure.

2. TEST GENERATION STRATEGY

The overall goal is to develop a test generation strategy that will produce consistently high quality characterization and production type tests for a wide range of devices in a cost effective manner.

2.1 What are the Obstacles?

Integrated circuit pin counts and operating frequencies continue to rise with advances in IC technologies. Test systems that meet these evolving testing requirements, and test program development time on those systems are becoming increasingly expensive. It is in the interest of IC design and manufacturing organizations to automate the generation of IC test programs for the purpose of saving time in the generation and maintenance of these test programs. Some ATE manufacturers are beginning to offer automatic TPG software, but a TPG that is conversant with in-house CAD tools is preferable. Also, a leading edge IC design and manufacturing organization can not afford to be tied to any one specific ATE vendor. ATE system vendors are independently developing hardware and software systems but there are no existing standards[1].

Ideally, an IC design and manufacturing organization would like to have a TPG which would automatically generate test programs suitable for every kind of IC device, from CAD tool output. The debugging facilities on the target test system would direct the user back to TPG

---

1. EDIF[11] and TISSS[12] represent efforts to develop standards.

parameters, thereby reducing the amount of study required for test engineers to use new test systems. The test program would be hidden from the user, the way machine language is hidden from a high level language programmer. Although systems of this type are emerging, they can not foreclose the possibility of accessing the test program for the following reasons:

- While testing requirements for the majority of devices are predictable, new technologies and designs bring about requirements which are either unforeseen or too unique to be incorporated in an automatic TPG.
- The TPG can not always make the final determination as to whether a particular type of test applicable to a given DUT pin.
- Power supply variations that occur due to the fluctuating power requirements of the DUT and the reaction time of the test system power supplies may influence test results.
- Waveform distortion that occurs due to impedance mismatches between the DUT, interface hardware, and the test system, affects test results.
- Long term variations of DUT parameters due to changes on the processing line affect test results.
- The target ATE debugging facilities refer to the test program, not to TPG parameters.
- The test program may be destined for an environment removed from that of the TPG, when second sourcing a device for instance.

The penalty for ignoring these factors is the introduction of measurement error, whose magnitude may not be tolerable. If, as expected, the user must eventually relate to the test program, then the structure of that program and its relationship to the TPG user interface will ultimately affect productivity.

2.2 Approaching the Problem

The conventional test generation approach, as depicted in Figure 1, is that the TPG generates a test program based on information supplied by CAD tools and users or both. The emphasis of this work is on the test program and the effect it has on the overall test program generation strategy. Figure 2 illustrates the main feature of the proposed variation on the conventional approach: the test program is divided into *device-dependent* and *general* files. The following sections offer some programming objectives and guidelines and a finer grained definition of the general test program structure. The concepts outlined in these sections are embodied in GENTEST, the author's implementation of a general test program in Advantest's ATL.

3. GENERAL TEST PROGRAM OBJECTIVES

A general test program should meet the following objectives:

- *Flexibility*: it must address the need to test a wide range of devices.
- *Timely Program Development*: it should consist of small source modules with reasonable compilation times and debugging hooks.
- *Provide Characterization*: it must be able to gather numeric data, a time consuming task.
- *Uniform Output for Data Analysis*: the data output of different test programs must be consistent enough to allow general data processing routines to extract pertinent information and prepare it for statistical analysis.
- *Provide Production Testing*: it must be able to classify the DUT as quickly as possible.
- *Ease of Program Operation*: it should have a standard operator interface.
- *Ease of Program Maintainability*: it should have a standard test program format and help reduce the proliferation of test program versions.

- *Correlation between the TPG User Interface and Test Program Structure* : TPGs are as yet not capable of producing a completed test program from CAD tool output except for a restricted class of devices. This implies the necessity of user intervention. To enhance productivity, the TPG User interface should have a readily discernible correlation to the test program structure.

- *Feedback to TPG*: the TPG must stay abreast of technology and associated testing techniques. Newly developed test techniques should find their way back to the TPG to insure continued broad-based applicability of the test programs generated by the TPG.

It is recognized that some of these objectives conflict. It is my intention that the test program structure and programming guidelines should provide a satisfactory resolution to these conflicts.

4. PROGRAMMING GUIDELINES

To encourage correspondence between the TPG user interface and the test program structure, it is proposed that the general IC test program structure acquire some of the features associated with TPGs such as a data base, templates, and tables. In the general IC test program structure, the data base is broken up into files which are inserted into the test program at compilation time by a compiler preprocessor. The test language macro is used to define individual tests (templates are used to perform that function, among others, in the TPG). Tables are constructed by the use of arrays, data structures, and macro calls.

Program structure symmetry, if not too costly, is preserved in the hope of revealing the systematic approach behind the test generation strategy. This symmetry, however, may elude the casual observer since almost every rule has its exceptions. For this reason I offer the following guidelines, the observance of which has resulted in the GENTEST structure.

General code is located in subroutines. The exception is the insert file INITGEN which

contains some generally useful definitions.

Device dependent code is located in insert files. The exceptions are the scramble table (the file SOCKET in the case of GENTEST) and the MAIN program. The default program flow, determined by MAIN, is adequate for most devices. For the remaining devices the flow is easily altered.

Insert files of the four types (initialization, set-up, specification, and macro call) are present in each module, if their contents can not be generalized. Initialization, in this context, refers to constants and variables used by more than one subroutine. The set-up files are so called because they set up the test system resources, that is, they set test system driver and comparator levels, timing generator offsets, and assign these resources to specific pins. The specification files contain the specification values against which the performance of the DUT is to be measured. The macro call files refer to the actual execution of individual tests. The exception is the FUNCTIONAL and FREQUENCY test module. The file VECLDS does not fit any of the four insert file types. It contains an array of vector load names. Also, there is no FQSPEC file even though the frequency tests are compared to a specification which can not be generalized. The frequency specification is in the INITDEV file because it must be shared by the AC test module.

All subroutines belonging to a module share the insert files belonging to that module. All test subroutines also share the insert files INITDEV and INITGEN. There are six modules: MAIN, PERFORMANCE BOARD test, CONTACT and DIODE test, FUNCTIONAL and FREQUENCY test, AC test, and DC test. The MAIN module is the exception. It controls test program flow, determines the net result of testing, and contains information which affects all other modules.

The guidelines so far, have been presented at the file level. Additional soft constraints can be placed upon file contents to facilitate comprehension and increase program flexibility and

execution speed.

The first of these additional constraints requires evaluating the relative worth of program attributes. Program execution speed and functionality are given preference over modularity. Modularity is given preference over symmetry. In the production subroutines, execution speed reigns supreme whereas in the characterization subroutines functionality dominates.

The second constraint is that the event sequence followed by the subroutines be clearly delineated. Each test subroutine roughly follows this sequence: initialize variables, define macros, power up the device, set up the test system, calibrate, execute tests, evaluate results, and set categories. Variations on this theme arise from the need to loop through some portion of these activities and the need to decide whether calibration is necessary. A pictorial representation is found in Figure 10.

The third constraint is to break up the test system set-up information into two components: one which will initialize test system resources and another which needs to be invoked with each repetition or invocation of a test. This is a time saving device used since the number of repetitions or invocations may grow quite large.

The fourth constraint is to put the information contained in the insert files in tabular form whenever possible. This makes it easier to locate pertinent information while debugging the test program or DUT and lends itself to tester independence.

The fifth and last constraint is to reduce macros to the lowest common denominator, bearing in mind the effect on test execution time. A case in point is the AC test. Instead of having separate macros to perform set-up, hold, assertion, negation, and tristate timing tests, there is one macro which performs all of these tests. The benefit is that the user can observe immediately that these tests are performed in a similar manner. Differences are easily spotted by comparing tabularly arranged macro call arguments.

5. GENTEST STRUCTURE

5.1 Test Language Dependency

To what extent is this structure system-independent? This question may be answered indirectly. Advantest ATL (*Advanced Test Language*), the language used to implement GENTEST, is a FORTRAN based language. The features of ATL that form the backbone of the GENTEST structure are the following:

- *External subroutines:* these are necessary to allow incremental compilation of the test program. This keeps compile time reasonable during program development.

- *The ATL INSERT directive:* this, or an equivalent thereof, is necessary to incorporate device-dependent information into test subroutines. This is the mechanism by which the sharing of device-dependent code takes place. The program statement

  INSERT FILENAME

  causes the translator to replace it with the contents of the source file FILENAME.

- *MACRO definitions:* these are desirable as a method of defining individual tests because they have no runtime overhead and and because they can be redefined.

- *Global variables:* these are necessary to pass information between external subroutines.

5.2 Test Routine Structure

The proposed structure is that of Figure 2. This is the same as in Figure 1 except that the test program source files are separated into files that contain *device-dependent* and *general* code. The device-dependent files serve as data for the general files, most of which are subroutines that perform incompatible tasks such as characterization and production type testing.

- The MAIN program controls program flow, summarizes test results, and categorizes devices.

- Menu selection of options is provided.

- A minimum number of subroutines is provided. Each subroutine performs one of the following test types: performance board, contact, diode, functional, frequency, AC, and DC. The frequency, AC, and DC routines each have a production and characterization version. The frequency characterization test is further broken down into a linear and binary search.

- Each of the previously mentioned subroutines contains individual test descriptions in the form of macros.

- Where both production and characterization versions of a subroutine exist, the production subroutine contains the production macro and the characterization subroutine contains the characterization macro. The two macro versions are invoked via one identifier from a common insert file. Which macro is invoked depends on which subroutine is running.

- The subroutines are independent with respect to the test system set-up that is, all pins are disconnected at the beginning of each subroutine.

5.3 GENTEST Modules

On the test system, GENTEST is a collection of files residing in a directory. These files can be thought of as belonging to groups or modules related by function and shared information. The following is a brief description of these modules. Figures 4 and 5 may clarify relationships between files and modules.

MAIN Module

The MAIN program directs the test program flow according to switches set via the subroutine GMENU and categorizes the DUT by summarizing test results passed back by the external subroutines. The file SOCKET contains the logical pin to hardware channel assignments. Only the program MAIN

and the vector loads must be compiled in the presence of a valid socket table. The insert files INITDEV and INITGEN contain device specific and general information which is to be shared between more than one external subroutine.

PERFORMANCE BOARD Test Module

These consist of the subroutine PRFBRD and the insert files PBRDCALL and WBRDCALL which contain macro calls to test the package and wafer performance board, respectively. The performance boards may be tested for shorts, leakage, relay functionality, and load resistors.

DIODE and CONTACT Test Module

The tests are performed by the subroutines DIODE and PLANE. Since these tests are similar, they share the utility subroutine SHOP, the insert file DIODSU, and two vector loads called DIODPAK and DIODWAF, to test packages or wafers. PLANE allows the operator to determine if contact has been made to the wafer under test. DIODE verifies the functionality of the P and N protection diodes.

FUNCTIONAL and FREQUENCY Test Module

There are two functional test subroutines: FNC and FNP for characterization and production respectively, and three frequency tests: FQBC and FQLC, for characterization, and FQP for production. FQBC and FQLC perform binary and linear searches respectively. Multiple categories are available to classify the DUT with respect to operating frequency. The associated insert files are FCALL, FSU, and VECLDS.

AC Test Module

There are two AC subroutines: ACC, for characterization, and ACP, for production. These subroutines will measure assertion, negation, tristate, set-up, and hold times as well as derived measurements such as pulse width or rise and fall time. Correction factors are incorporated before judgment is passed on the results. Multiple categories are available to classify the DUT. The associated insert files are ACCALL, ACSU, and ACSPEC.

DC Test Module

There are two DC subroutines: DCC, for characterization, and DCP for production. These subroutines will test for VIH, VIL, VOH, VOL, IOH, IOL, IOZH, IOZL, IIH, IIL, and dynamic and static IDD. The associated insert files are DCCALL, DCSU, and DCSPEC.

5.4 File Naming Conventions

The GENTEST naming conventions are as follows:

- Files associated with the AC (timing) tests *begin* with the letters "AC".
- Files associated with the DC (voltage and current) tests *begin* with the letters "DC".
- Files containing set-up (test system resource allocation) code *end* with the letters "SU".
- Files containing macro calls *end* with the letters "CALL".
- Files containing DUT specification data *end* with the letters "SPEC".
- Production external subroutine names *end* with the letter "P".
- Characterization external subroutine names *end* with the letter "C".
- Performance board test related files *contain* the letters "BRD".

5.5 File Types

The file types are as follows:

- *Command file*: These files are provided to initialize test set conditions and restore default switch settings.
- *Program*: GENTEST contains only one program: MAIN.
- *External subroutine*: External subroutines are kept in separate files and are invoked using the CALL statement as opposed to internal subroutines which are invoked using the GOSUB statement. External subroutine files contain macro definitions.
- *Insert file*: An insert file starts with the comment "GENTEST INSERT FILE FILENAME" by convention. A file of this type is automatically inserted into an external subroutine or program at translation time by the statement "INSERT FILENAME".
- *Socket table*: GENTEST contains only one socket table, the file SOCKET.
- *Vector load*: Vector loads are supplied by the user. Three special purpose vector loads are required: DIODPAK, DIODWAF, and DCLOOP.

The MOST XII operating system command "**ALIST FILENAME**" will list the first line of the file FILENAME. This is useful for identifying file types and the program. If all GENTEST programs use the name SOCKET for example, then ALIST SOCKET can be used to identify the device, provided that the first line of the file SOCKET is appropriately commented.

5.6 File Contents

Figures 6, 7, and 8 show the file contents of the core GENTEST files. The listing below includes those files and some files which are not part of the test program, but perform some supporting functions.

ACC	*external subroutine* — AC (timing) test for characterization.
ACCALL	*insert file* — this file, which contains AC (timing) test macro calls, is inserted into the external subroutines ACC and ACP.
ACP	*external subroutine* — AC (timing) test for production.
ACSPEC	*insert file* — AC (timing) specifications and correction factors; used by ACC and ACP.
ACSU	*insert file* — test system set-up (resource allocation) for ACC and ACP.
CHANNL	*external subroutine* — converts software pin lists to hardware channel lists and vice versa.
DCC	*external subroutine* — DC (voltage and current) test for characterization.
DCCALL	*insert file* — contains DC (voltage and current) test macro calls; used by DCC and DCP.
DCP	*external subroutine* — DC (voltage and current) test for production.
DCSPEC	*insert file* — contains DC (voltage and current) specifications; used by DCC and DCP.
DCSU	*insert file* — test system set-up (resource allocation) for DCC and DCP.
DIODE	*external subroutine* — P and N protection diode test.
DIODSU	*insert file* — test system set-up (resource allocation) for DIODE and PLANE; assigns test system resources.
FCALL	*insert file* — contains functional test macro calls; used by FNC, FNP, FQBC, FQLC, and FQP.
FNC	*external subroutine* — functional characterization test.

FNP — *external subroutine* — functional production test.

FSU — *insert file* — test system set-up (resource allocation) for FNC, FNP, FQBC, FQLC, and FQP.

FQBC — *external subroutine* — frequency binary search for characterization.

FQLC — *external subroutine* — frequency linear search for characterization.

FQP — *external subroutine* — frequency test for production.

GLOBS — *external subroutine* — used to initialize the global variables which are used by the GENTEST program. The values stored in these variables normally survive from one test system user to the next. Since these variables affect program behavior, they need to be initialized.

GMENU — *external subroutine* — used to display and set GENTEST program options.

INITDEV — *insert file* — user supplied device-dependent information: initialization of pinlists and variables. It contains variables that control the operating frequency and the power supply voltages at which the external subroutines are executed, pass category definition, and all variables that are affected by test mode; it is used by MAIN and all external subroutines except SHOP and GMENU.

INITGEN — *insert file* — contains device independent definitions: fail categories, function keys, log keys, and action selectors, timing generator numbers, and various variables shared by MAIN and all external subroutines except SHOP.

MAIN — *program* — controls program flow by way of function key switches and external subroutine calls and interprets categories set by external subroutines to arrive at a summarizing categorization of the DUT.

PAK — *command file* — should be executed in test mode, at the outset of each session, to initialize test system for package testing.

PBRDCALL — *insert file* — contains package performance board test macro calls; inserted into PRFBRD.

PLANE — *external subroutine* — checks for contact between the wafer and the probes.

PRFBRD — *external subroutine* — performance board check; if the performance board identification, that is, the control word (CW) on the performance board, matches the variable PBRDID then the macros in PBRDCALL are invoked, if the performance board identification matches the variable WBRDID then the macros in WBRDCALL are invoked. The variables PBRDID and/or WBRDID are defined in the file INITDEV.

SETUP1 — *command file* — contains function key, log key, and action selector switch settings saved by GMENU when executed on test head 1.

SETUP2 — *command file* — contains function key, log key, and action selector switch settings saved by GMENU when executed on test head 2.

SHOP — *external subroutine* — checks for shorts, opens, and high impedance contacts; called by PLANE and DIODE.

SOCKET — *socket table* — logical pin to hardware channel scramble table.

VECLDS — *insert file* — array of vector load names used by FNC, FNP, FQBC, FQLC and FQP.

WAF — *command file* — should be executed in test mode, at the outset of each session, to initialize test system for wafer testing.

WBRDCALL — *insert file* — wafer performance board test macro calls; inserted into

PRFBRD.

6. RESULT

The purpose of GENTEST is to reduce test program development and maintenance time in the face of uncertainty about the requirements of future designs and technologies. The test program for a specific device may be generated either automatically, as the output of an automatic TPG, or manually, by filling out the database portion of the GENTEST structure. Either process is faster by an order of magnitude than manually generating a test program without the aid of a TPG or GENTEST structure. It is difficult to be precise about the gains made through the application of these methods because many factors influence the results. Factors that influence the amount of time it takes to generate a test program are the experience of the test engineer, the amount of time allotted to the test engineer on the ATE, the complexity of the DUT, the requirements of the DUT (do they tax the capabilities of the ATE?), the availability of information about the DUT, and the applicability of previously developed testing methods to the DUT. Nonetheless, the author's subjective observations are that it may take a test engineer three months to a year and a half to manually generate a test program. The second GENTEST program, requiring the provision of device-dependent information only, was generated in a week. TPG2 can generate a test program in less than an hour, provided the manual data entry is minimal, that is, the data is extracted from CAD tools.

6.1 GENTEST Attributes

The contribution that GENTEST can make as the output of a TPG is difficult to quantify, but the following sections describing GENTEST attributes should explain why the author believes that the contribution is a positive one. Each of the following sections views GENTEST with a different goal in mind.

6.1.1 Test Program Generation

When a test program generator is used to generate an IC test program of the GENTEST structure, the test generation process benefits because the TPG can provide services which GENTEST can not provide for itself. For example, a TPG can check data types and limits, offer a simpler user interface, add a level of flexibility, and extract information required for documentation. GENTEST, when generated by a TPG, can improve the test generation process because it provides production and characterization tests with one data set, it increases the number of devices for which the TPG is a useful tool, it provides a feedback path of testing methods to the TPG, and it allows a TPG output processor to be implemented incrementally. In short, the TPG generating a test program employing the GENTEST structure is more powerful than either would be by itself. If GENTEST is generated by an automatic TPG, it will not greatly affect the test program development time for the majority of devices whose needs are predictable, considering the current automatic test generation methods. It will expedite the program development for unusual devices and the program maintenance required for all devices. It is the author's belief that the GENTEST structure attains the previously stated objectives and that compromises, though unavoidable, are minimal.

Although GENTEST can be used to generate a test program manually, the view is taken that a TPG will ultimately generate the first draft of a test program. Being denied perfection, TPGs and test programs are expected to be under development at all times. The interface between a TPG and a test program should therefore ensure the maximum independence of the two. This interface is embodied in the GENTEST INSERT files which contain device-dependent information described by a subset of the target system programming language, that is Advantest's ATL. The bulk of these files fall under the headings of initialization, test system set-up (resource allocation), macro calls (test execution), and device specifications.

This information is then utilized by the general test routines which may be generated by the TPG or copied from a standard library to a user's directory. This approach has several ramifications:

- The TPG can give a user the opportunity to save alterations made to general source files when regenerating a test program, that is, the TPG can recognize when a general file has been altered and choose to overwrite the device-dependent files only. This is deemed necessary because, even though most test programs perform the same types of tests, there are, by necessity, variations to the manner in which these tests are executed. Altered tests can be marked non-standard by the TPG.

- A user, familiar with the specification of the interface between the TPG and the test program or the format and nature of the device-dependent information if preferred, is free to develop reusable routines to fill needs not addressed by the TPG. If deemed general enough, these can find their way back to the TPG to be offered to other users.

- Since the device-dependent information is expressed in a *subset* of the target machine language, it may be easier to provide a read-back mechanism from the test program to the TPG. This is done to allow test program regeneration on the same test system or test program generation on a different test system.

The GENTEST program also meets some other concerns. Mechanisms exist for the sharing of AC and DC test results between external subroutines, in case these are needed for runtime adjustments. The functional and AC modules allow multiple set-ups in case one set of test system resource allocations will not suffice for all vector loads. The frequency and AC tests will check against multiple sets of device specifications for the separate binning of fast and slow devices. Guard-banding for the frequency, AC, and DC tests is provided for production testing at where the value of the guard-band varies depending on what stage in the production process the testing is being done. The shared database eliminates the previous

redundancy of effort required to produce programs for characterization and production.

The development of a standard format test program is encouraged since this is the default configuration of GENTEST whereas most of the variations required to test unusual or leading edge devices can be incorporated with relatively little effort.

6.1.2 Debugging and Fine Tuning

A major purpose for the existence of GENTEST is to relieve the user of the task of having to debug the test program. A major function of the TPG is to extract accurate data from CAD tools to drive the test program. Although much progress has been made to achieve these goals, ultimately the user must insure that the DUT and not the test system or a test program error is causing a test to pass or fail. The GENTEST program helps by providing direct access to failing tests where possible and diagnostic output to complement test system tools. The device-dependent information which is frequently altered during the debugging process is accessible by file name in concentrated and, when possible, tabular form to avoid time consuming searches.

6.1.3 Operation and Maintenance

Operation and maintenance are of largest concern in the production environment. The features that aid in debugging and fine tuning are helpful during program maintenance as well. Device specifications, guard-band values, correction factors, and device binning are easily altered with or without the aid of a TPG. The standard test program format is desirable from the point of view of the production test engineer who must maintain many such programs. This format suffices for a great number of devices thereby encouraging its use. The fact that for each production test there is a parallel characterization test is sometimes helpful when troubleshooting a test program.

6.1.4 Data Collection

GENTEST produces data in a format that is easily converted into a format recognizable by a data analysis tool such as the UNIX® tool "S". Selective gathering of data may be done via program switches set from the menu. When gathering data on a large number of devices the user may instruct the program to collect AC data only if the DUT passes the functional and frequency tests.

6.2 GENTEST Applied

GENTEST has been used to test two devices: the WE® 32206 Math Acceleration Unit[13] and a Wide Band Quad Receiver/Driver/D Flip-Flop, which is not commercially available. These two devices occupy opposite ends of the spectrum in terms of transistor count. The first is a CMOS and the second an NMOS device. Figures 11 and 12 show the menu that is displayed on the CRT by the subroutine GMENU. Test program output depends on which menu items are selected. Figures 13 through 20 show the type of data generated by the test program in the characterization mode. Figure 20 is the output format generated by choosing to run the diode, functional, frequency, AC, and DC tests without invoking any of the options associated with those tests.

Both devices were subjected to functional, frequency, AC, and DC tests. The diode test was applicable to the CMOS but not to the high speed NMOS device. Test results appeared reasonable when compared to oscilloscope readings. The effects of the test system transmission line, which connects the DUT to the test system pin electronics, were noticeable in the AC tests but were accounted for by incorporating offsets.

The tests for both devices were programmed by inserting code into the device-dependent files only with two exceptions. The first line of the MAIN program was altered to incorporate the name of the DUT and, because the production test section of MAIN calls the subroutine DIODE by default, that call was removed in the case of the high speed NMOS device.

6.3 The ATE Vendor and GENTEST

A program utilizing the GENTEST structure could be supplied by an ATE vendor. This would allow an IC design and manufacturing organization to write a TPG output processor for a specific test system incrementally. Each increment lends additional flexibility to the TPG so that it can generate test programs for a wider variety of devices:

- The TPG generates the insert files (device-dependent code) only. The user can change test program flow by editing the source file MAIN.

- The TPG generates the program flow portion of the file MAIN.

- The TPG generates unique label names for the GENTEST subroutines (general code).

- The TPG generates application specific macros for the GENTEST subroutines (general code).

The last item allows the user to compile MAIN with a compiler option whose effect is the following: instead of having a MAIN program call external subroutines, the external subroutines are treated as insert files thus creating one large object file. The large MAIN program object file executes faster than a MAIN program object file that calls external subroutines but takes much longer to compile.

6.4 Summary

The author has written a general test program, named GENTEST, based on established testing practices and procedures. GENTEST has been used to successfully test the WE® 32206 Math Acceleration Unit and a Wide Band Quad Receiver/Driver/D Flip-Flop. GENTEST programs are currently generated by manually entering the code required in the device-dependent files. Observation confirms that using this method is an order of magnitude improvement over manually generating an entire test program. This improvement is manifest in the amount of time it takes to generate and debug a test program, and in the quality and

consistency of the test programs generated. Most of the methods used to achieve speedy program execution necessary for production have been incorporated in the general code. Device characterization is available at the flick of a switch. Parallelism between production and characterization routines allows cross-checking test results. This parallelism may be forfeited for still faster execution time, if desired, by substituting a tight functional test for several parametric tests.

A TPG could generate a GENTEST type structure to the benefit of both. GENTEST would benefit because the TPG can perform functions like:

- Gathering CAD data.
- Checking data types and limits.
- Providing a friendlier user interface.

The TPG could benefit by employing the GENTEST structure because:

- The TPG provides production and characterization tests with one data set.
- The number of devices for which the TPG is a useful tool, is increased.
- The feedback path of testing methods to the TPG is improved.
- A TPG output processor can be implemented incrementally.

REFERENCES

1. 1981. "ATLAS Test Language." *IEEE Std. 416-1981.* IEEE, New York.

2. Watson, I. M., J. A. Newkirk, R. Mathews, and D. B. Boyle. 1982. "ICTEST: A Unified System for Functional Testing and Simulation of Digital ICs." *IEEE Digest of Papers 1982 International Test Conference* pp. 499-502.

3. Syed, I., and N. Rose. 1982. "Automated Generation of Device Test Software." *IEEE Digest of Papers 1982 International Test Conference* pp. 514-521.

4. Davidson, R. P. 1976. "A Tester Independent Program Language." *IEEE 1976 Semiconductor Test Symposium* pp. 76-77.

5. Grinthal, E. T. 1978. AT&T Bell Laboratories, private communication.

6. Bednarczyk, P. J. 1978. AT&T Bell Laboratories, private communication.

7. Millheim, K. E. 1982. AT&T Bell Laboratories, private communication.

8. Ahrens, D. P., P. J. Bednarczyk, D. L. Denburg, and R. M. Robertson. 1984. "TPG2 - An Automatic Test Program Generator for Custom ICs." *IEEE International Test Conference 1984 Proceedings* pp. 762-767.

9. Bednarczyk, P. J., D. L. Denburg, and R. M. Robertson. 1987. "TPG2 - An Automatic Test Program Generation System for Design and Manufacture of ASICs." *IEEE International Test Conference 1987 Proceedings* pp. 96-104.

10. Gordon, R. L. 1984. AT&T Bell Laboratories, private communication.

11. Eurich, J. P. 1986. "A Tutorial Introduction to the Electronic Design Interchange Format." *IEEE Design Automation Conference 1986 Proceedings* pp. 327-333.

12. Falkenstrom, L. J., D. Keezer, A. Patterson, R. M. Rolfe, and J. Wolcott. 1985. "Tester Independent Support Software System (TISSS)." *IEEE International Test Conference 1985*

Proceedings pp. 685-690.

13. *AT&T UNIX® Microsystem Catalog.* October, 1986.

14. Stevens, A. K. 1986. *Introduction to Component Testing.* Addison-Wesley Publishing Co.

15. Hickling, R. L. 1983. "Tester Independent Problem Representation and Tester Dependent Program Generation." *IEEE International Test Conference 1983 Proceedings* pp. 476-482.

16. Okamoto, T., H. Shibata, and K. Kinoshita. 1983. "Design of High Level Test Language for Digital LSI." *IEEE International Test Conference 1983 Proceedings* pp. 508-513.

17. Wilber, J. G. 1985. "Enhancing Device Test Programming Productivity: The CATalyst Automated Test Program Generator." *IEEE International Test Conference 1985 Proceedings.* pp. 252-262.

18. Muranaga, K., K. Sakurada, Y. and Oikawa. 1985. "Language Independent Test Generation (LITG)." *IEEE International Test Conference 1985 Proceedings.* pp. 263-270.

19. Contini, M. 1985. "The AUTOPAL Test Process." *IEEE International Test Conference 1985 Proceedings.* pp. 279-285.

20. Chang, J. M., W. T. Krakow, and G. Kedem. 1987. "A Generic Test Program Translator (GTPT) for Tester-per-pin ATE." *IEEE International Test Conference 1987 Proceedings* pp. 1044-1051.

7. APPENDIX: Glossary of Terms

AC test	Waveform timing measurement.
ATL	Acronym for Advantest's Advanced Test Language, an extended FORTRAN type language.
assertion	0 to 1 transition for positive logic. 1 to 0 transition for negative logic.
assertion test	AC test that measures propagation delay for DUT output assertion.
characterization test	A test that reports discrete measured values.
DC test	Steady state voltage or current measurement.
derived test	AC test measurement such as pulse width or duty cycle that involves two edges and must therefore be derived from two previous measurements.
DUT	Acronym for Device Under Test.
external subroutine	Subroutines that are called by the CALL statement and return via the RETURN statement. An external subroutine resides in a separate file headed by the statement SUBR SUBROUTINENAME. No reference is made to a socket table because the subroutine inherits that table from the calling program.
dynamic IDD	DC test that measures the amount of current consumed by a device when active.
frequency test	Functional test performed at various frequencies to find limiting frequency or confirm operation at specified frequencies.
function key	A key on the test system control box whose position, on or off, can be determined by the test program.

functional test	Confirms that the DUT conforms to its truth table.
high threshold test	DC test. See VIH.
hold test	AC test that measures the amount of time an input signal must be held after a latch or reference point in order to be latched.
IIH	DC test that measures input leakage. The input to a buffer is pulled high (VIH). The leakage current is expected to stay within specification (IIH).
IIL	DC test that measures input leakage. The input to a buffer is pulled low (VIL). The leakage current is expected to stay within specification (IIL).
insert file	A file whose text is inserted into another file as part of the translation process. The ATL statement *INSERT FILENAME* precipitates this action.
internal subroutine	Subroutine that is called by the GOSUB statement and returns via the GOTO CONTINUE statement. It resides within the file from which it is called. If the contents of FILE1 reside in FILE2 because FILE2 contains the statement INSERT FILE1, subroutines in FILE1 are deemed internal to FILE2.
IOH	DC test that measures output current. The high DUT output is pulled low (VOL). The output current is expected to be better than or equal to specification (IOH).
IOL	DC test that measures output current. The low DUT output is pulled high (VOH). The output current is expected to be better than or equal to specification (IOL).

IOZH	DC test that measures output leakage current. The DUT output is tristated and then pulled high (VOZH). The leakage current is expected to stay within specification (IOZH).
IOZL	DC test that measures output leakage current. The DUT output is tristated and then pulled low (VOZL). The leakage current is expected to stay within specification (IOZL).
load board	Synonym for performance board.
loose functional test	The set-up for this type of functional test gives the device every benefit of doubt, that is, voltages and timing are favorably in excess of specification values.
low threshold test	DC test. See VIL.
module	A group of subroutine and insert files that share a common purpose.
negation	1 to 0 transition for positive logic. 0 to 1 transition for negative logic.
negation test	AC test that measures propagation delay time for DUT output waveform negation.
output processor	The portion of a TPG that converts tester-independent data into a test program for a specific test system.
parametric test	Either an AC or DC test.
performance board	A printed circuit board which serves as an interface between the test head and the DUT.
production test	Go-nogo test to insure device performs at specification.
S	UNIX® resident software for Data Analysis and Graphics written by Richard Becker and John Chambers.

scramble table	Synonym for socket table.
set-up	Synonym for test system resource allocation. Refers to the assignment of voltage levels, currents, waveform formats, and other parameters to individual pins prior to performing a test.
set-up test	AC test that measures the amount of time an input signal must be asserted before a latch or reference point in order to be latched.
socket table	A table, used by the program, that describes the correlation of test system hardware pins to logical pins. The logical pins are grouped by the programmer with respect to DUT pin types such as ADDRESS, DATA, INPUT, OUTPUT, I/O, etcetera.
static IDD	DC test that measures the amount of current consumed by a device when inactive.
subroutine	See internal or external subroutine.
switch	A bit that is used to govern the behavior of the test program. A switch may be turned on or off via software or hardware.
target	Used to denote the ATE for which a TPG is generating a test program.
test head	The part of the test system containing the electronics which stimulate the DUT and relay its response.
test set	A test system.
threshold	DC test that measures the borderline voltage at which the DUT latches the intended state. See VIH and VIL.
tight functional test	A functional test whose set-up is done strictly to device specification. One test of this type may replace several, if not all, AC and DC

	production tests.
tristate test	AC test that measures propagation delay time for DUT output waveform to tristate.
vector	One line of a DUT truth-table referred to as vector load.
VIH	DC test that measures the input threshold voltage. The logic 1 voltage, applied to DUT input(s), is varied to find the point at which the DUT first recognizes the voltage as a logic 1.
VIL	DC test that measures the input threshold voltage. The logic 0 voltage, applied to DUT input(s), is varied to find the point at which the DUT first recognizes the voltage as a logic 0.
VOH	DC test that measures high output voltage. The DUT output voltage is expected to remain above the specified minimum high level (VOH) while sinking the specified amount of current (IOH).
VOL	DC test that measures low output voltage. The DUT output voltage is expected to maintain the specified maximum low level (VOL) while sourcing the specified amount of current (IOL).

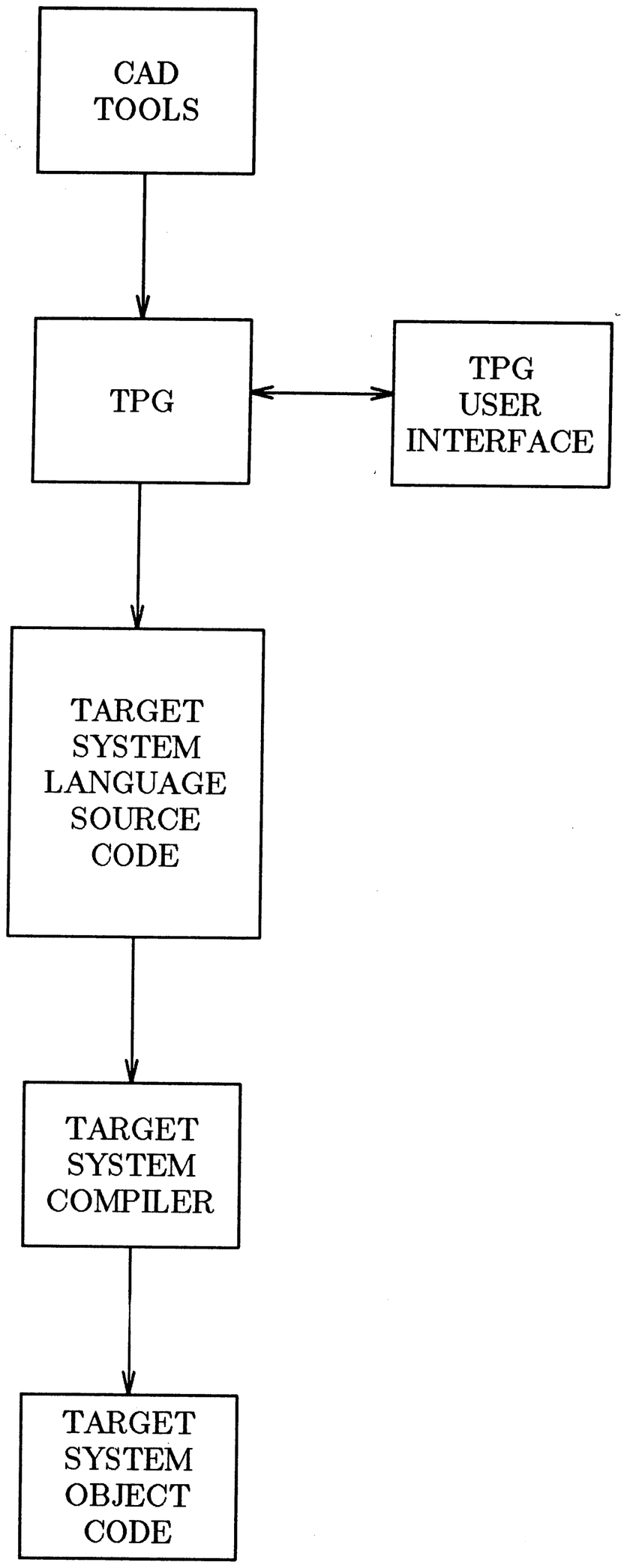

Figure 1. Current Test Program Generation Approach

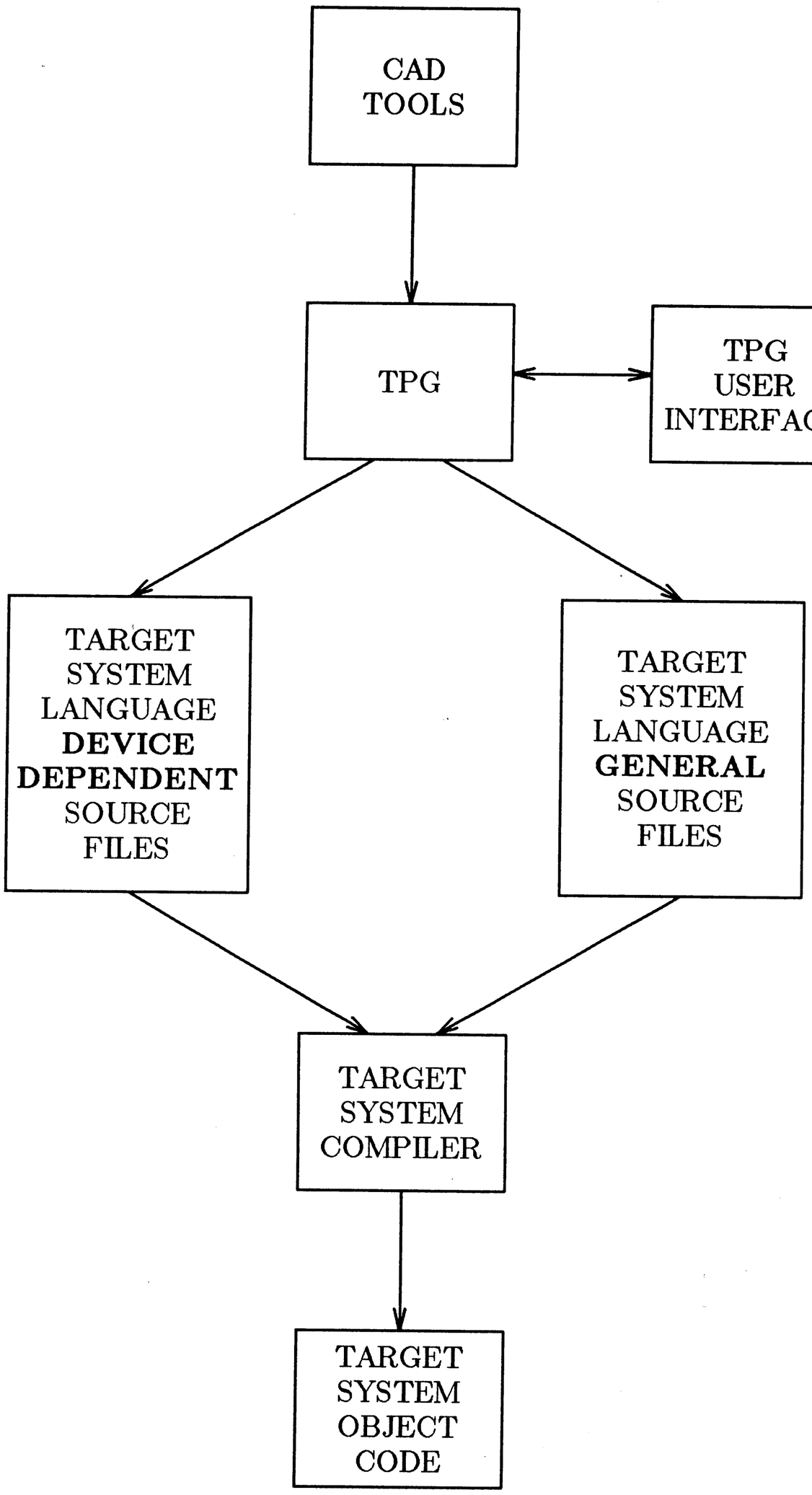

Figure 2. Proposed Test Program Generation Approach

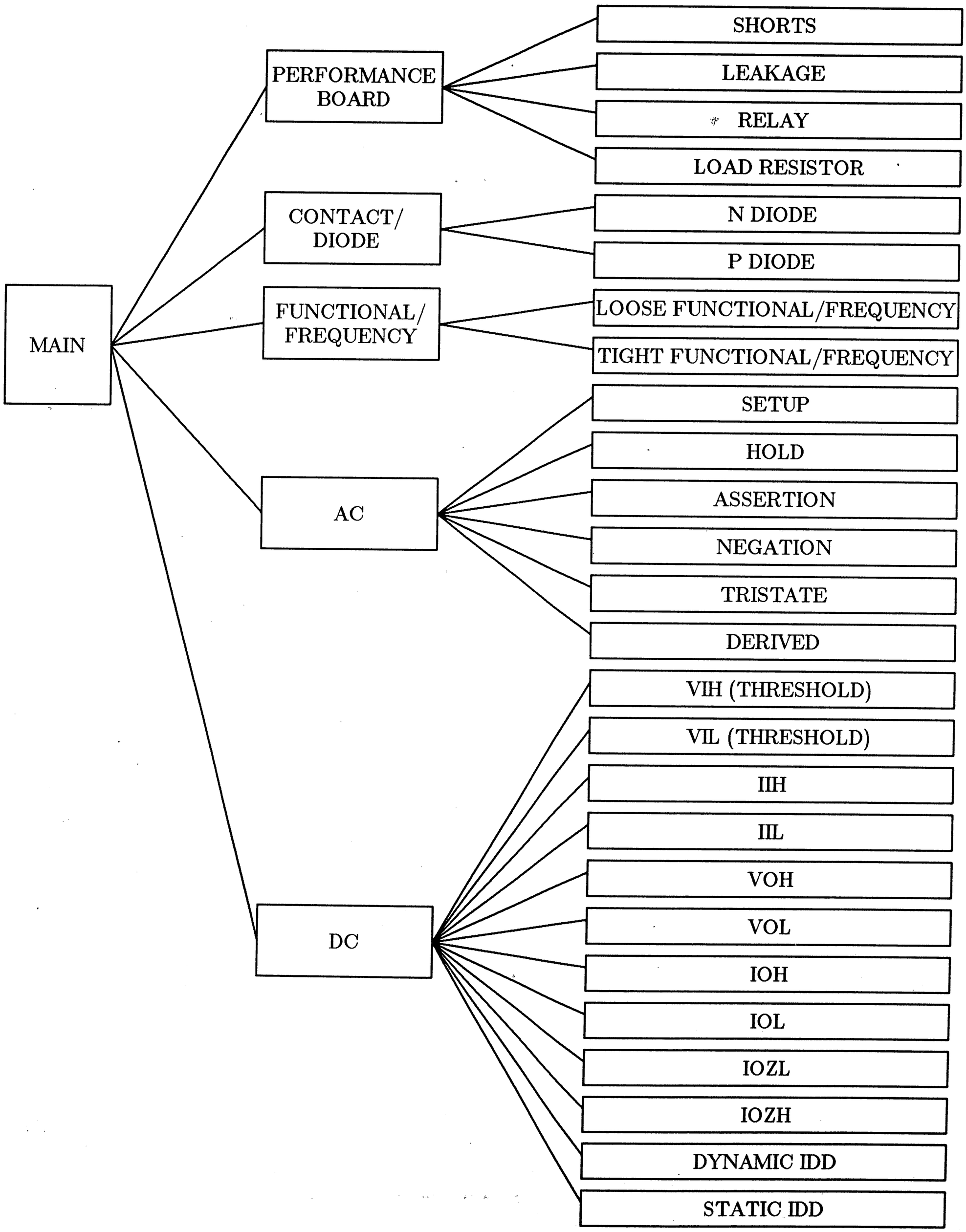

Figure 3. A Hierarchical Representation of Commonly Performed Tests

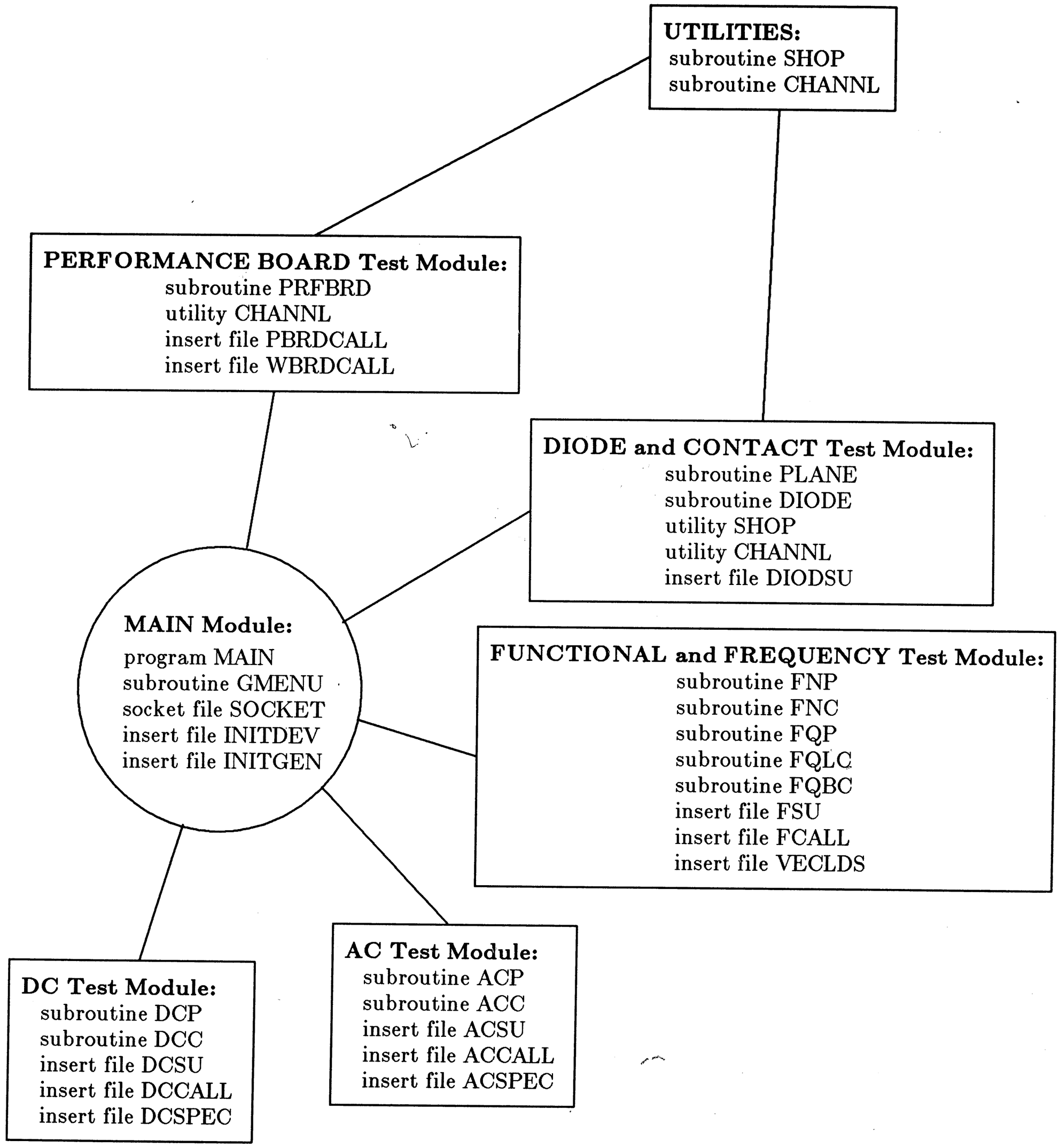

Figure 4. Interrelation of GENTEST Modules

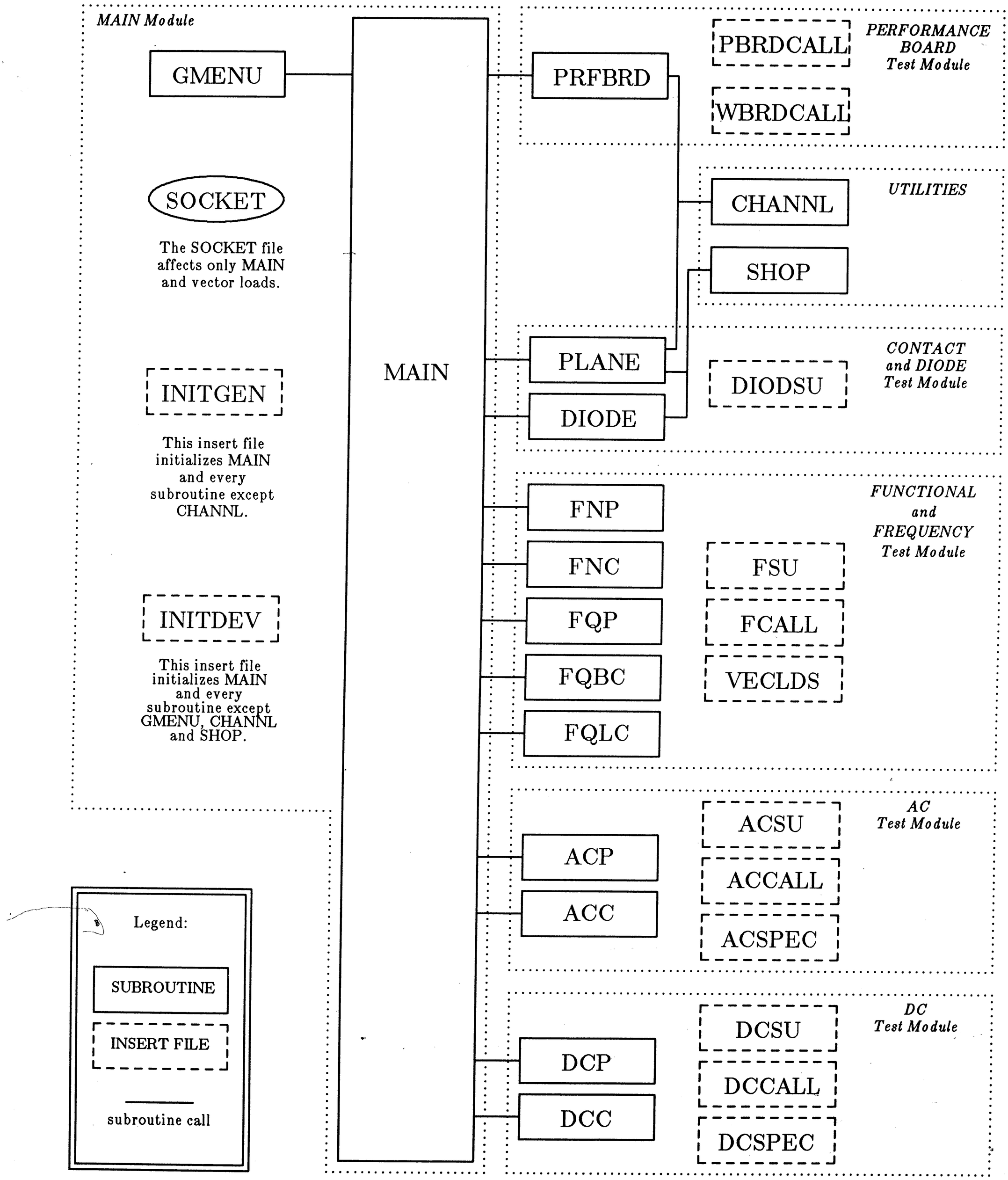

Figure 5. Interrelation of GENTEST Files

Insert Files

FILE INITGEN

preprocessor definitions
- terminal control sequences
- bit masks:
  - function keys
  - action selectors
  - fail categories

global variables
- timing generator definitions
- control box status bits
- miscellaneous

FILE INITDEV

global read-only storage
- pinlist definitions
- pass categories
- test periods:
  - functional
  - AC
  - DC
  - dynamic IDD
- power supply definitions:
  - VSS
  - VDD
- power supply values:
  - functional
  - frequency
  - AC
  - DC
- guard band values
- performance board ID

switch sensitive values
- test modes

global variables
- optional

FILES DIODSU
- FSU
- ACSU
- DCSU

test system resource allocation

FILES PBRDCALL
- FCALL
- ACCALL
- DCCALL

macro calls

FILE ACSPEC

timing specs
test description

FILE DCSPEC

voltage specifications
current specifications

FILE VECLDS

functional vector load names

Figure 6. File Contents: Insert Files

Main Program

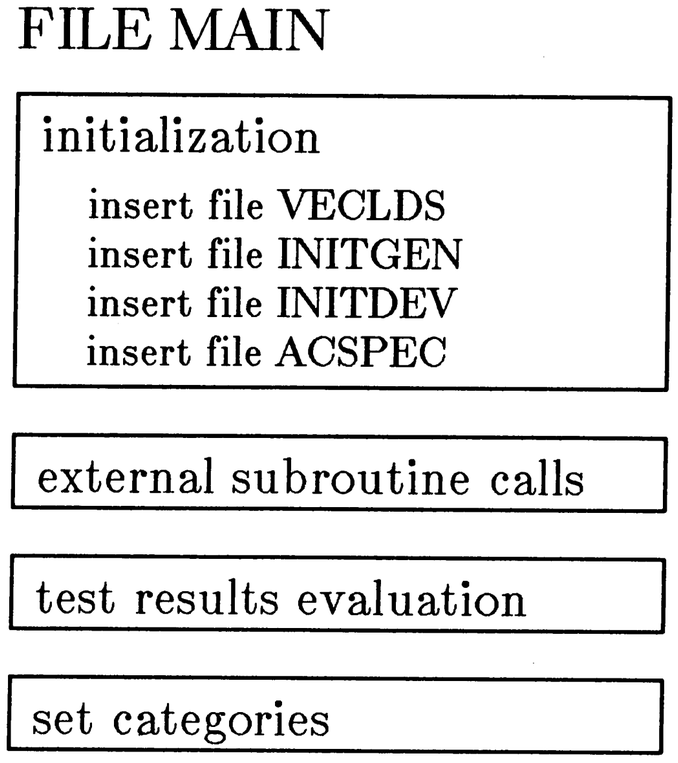

Figure 7. File Contents: MAIN Program

Subroutines

FILE GMENU

selection menu

FILE PRFBRD

performance board tests
- shorts
- leakage
- relays
- resistors

FILE PLANE

contact tests
- wafer contact
- contact resistance

FILE DIODE

diode tests
- P diode
- N diode

FILES FNP and FNC

functional tests

FILES FQP, FQLC, and FQBC

frequency tests
- pass/fail
- linear search
- binary search & clock skew

FILES ACP and ACC

timing tests
- setup
- hold
- assertion
- negation
- tristate
- derived

FILES DCP and DCC

voltage and current tests
- input threshold high
- input threshold low
- input leakage high
- input leakage low
- output leakage high
- output leakage low
- output tristate leakage high
- output tristate leakage low
- output voltage high
- output voltage low
- dynamic power supply current
- static power supply current

Figure 8. File Contents: Subroutines

MAIN Program Flow Chart

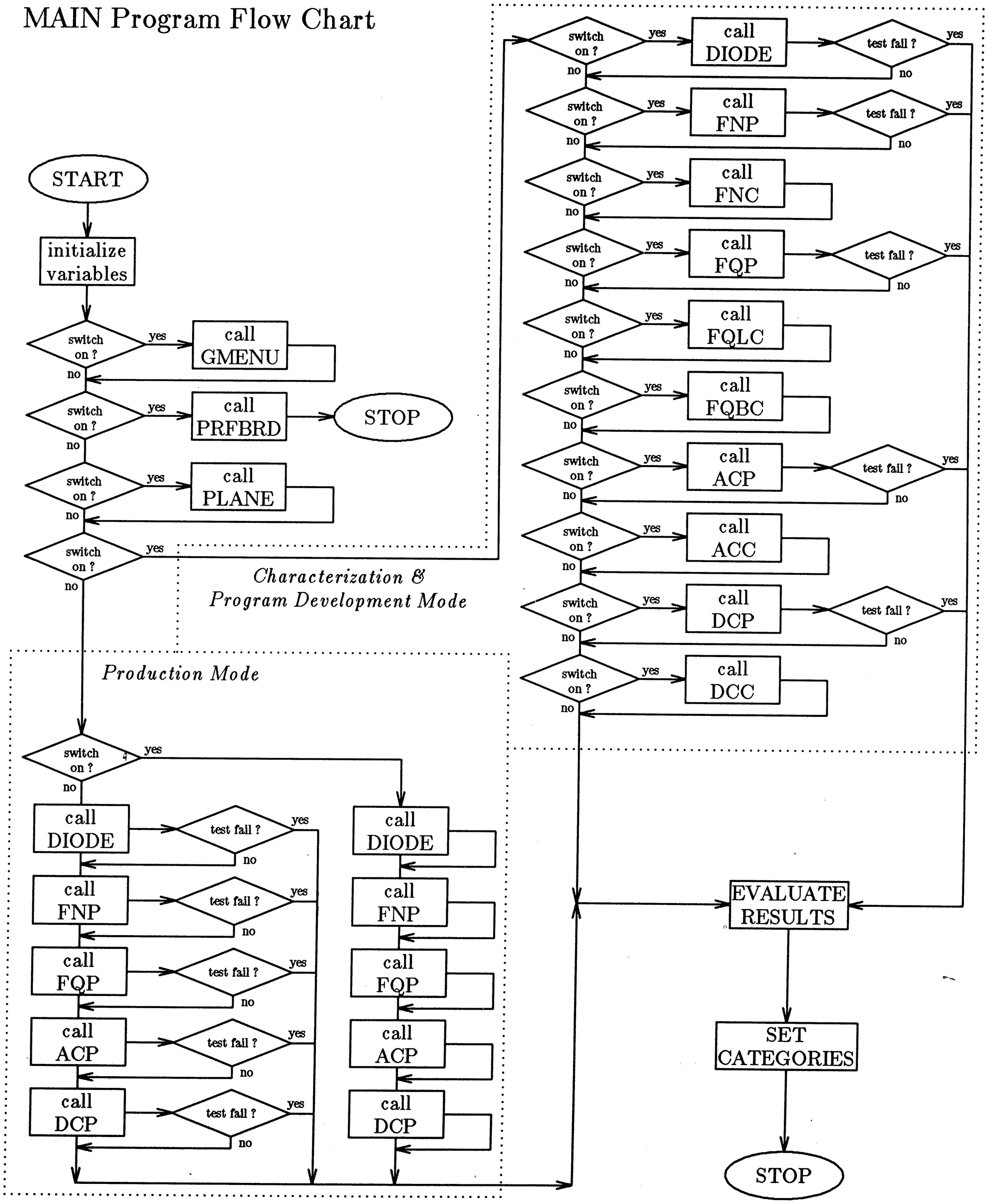

Figure 9. Flow Chart: MAIN Program

GENERAL TEST SUBROUTINE Flow Chart

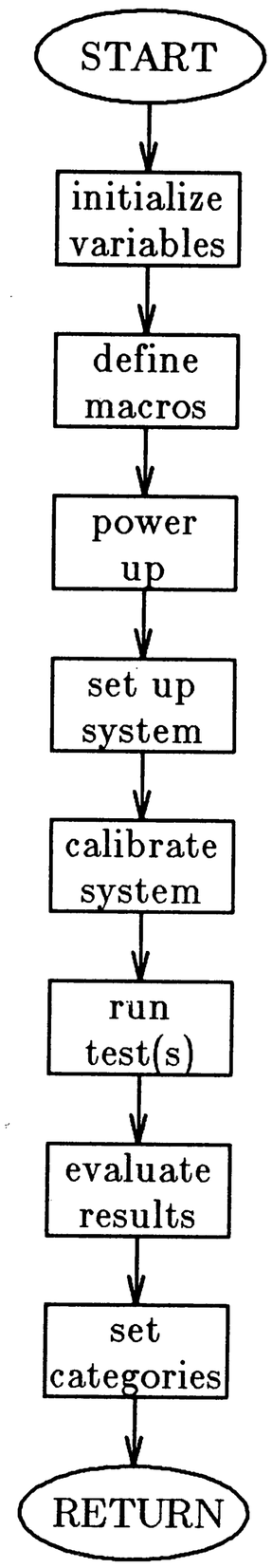

Figure 10. Flow Chart: Typical Test Subroutine

CHARACTERIZATION & PROGRAM DEVELOPMENT MENU

Type H or ? for help

```
 1  PERFORMANCE BOARD TEST
 2  CONTACT RESISTANCE TEST
 3  DIODE TEST
 4      show failing pins and channels
 5  FUNCTIONAL TEST production version
 6  FUNCTIONAL TEST
 7      show first failure locations
 8      show failure mode
 9  FREQUENCY TEST production version
10  FREQUENCY TEST LINEAR SEARCH
11      show frequency limitation of each vector load
12      show failure mode
13      show pass/fail histogram
14  FREQUENCY TEST BINARY SEARCH
15      show frequency limitation of each vector load
16      find worst case skew ( 4 phase clocked devices only )
17  AC TEST production version
18  AC TEST
19      show individual results
20      debug mode
21      set trigger ste. adrs.
22  DC TEST production version
23  DC TEST
24      show individual PASS results
25      show individual FAIL results
26  SPECIAL TEST
27  CHARACTERIZATION & PROGRAM DEVELOPMENT MODE
28  DISPLAY MENU
```

Figure 11. Subroutine GMENU: Characterization & Program Development Mode

PRODUCTION TEST MENU

Type H or ? for help

```
 1  FK1          Performance Board Test
 2  FK2          Contact Setup Test
 3  FK3          Collect Failing Vector Data
 4  FK8          Print Test Time
 5  FK9          Continue After Fail
 6  FK14         Characterization & Program Development Mode
 7  FK16         Display Menu
 8  PASS KEY     Display Passing Tests
 9  FAIL KEY     Display Failing Tests
10  MARK KEY     Enable Failing Pin Display: DIODE test
11  PAR  KEY     Enable Detailed Info: AC & DC tests
```

Figure 12. Subroutine GMENU: Production Mode

```
F1      = TEST 70001      Description: SAS    SU      Verification pin(s): SRDY0          Reference: CLK34  1,0
                          Vector Load: MAUA001        Window Address:    59 = hex   3B    Location:    50 = hex   32
                          Char offset = 0.000S        Prod offset = 0.000S
BCLK1 = <1>   96.500NS                                                                    Result =     3.500NS

F2      = TEST 70002      Description: SAS    HO      Verification pin(s): SRDY0          Reference: CLK34  1,0
                          Vector Load: MAUA001        Window Address:   155 = hex   9B    Location:   146 = hex   92
                          Char offset = 0.000S        Prod offset = 0.000S
BCLK1 = <1>   98.125NS                                                                    Result =    -1.875NS

F19     = TEST 70003      Description: DATA   TS      Verification pin(s): DATA           Reference: DSHAD0 1,0
                          Vector Load: MAUA005        Window Address:   247 = hex   F7    Location:   247 = hex   F7
                          Char offset = -6.200NS      Prod offset = 6.200NS
STRB1 = <1> **********    ERROR: PASSes at STRB1 = <1> 0.000S, the intended FAIL edge.

F20     = TEST 70004      Description: DATA   AS      Verification pin(s): DATA           Reference: DSHAD0 0,1
                          Vector Load: MAUA005        Window Address:   251 = hex   FB    Location:   251 = hex   FB
                          Char offset = -3.200NS      Prod offset = 3.200NS
STRB1 = <1> **********    ERROR: FAILs  at STRB1 = <1> 200.0NS, the intended PASS edge.

F41     = TEST 70005      Description: SRDY0  TS      Verification pin(s): SRDY0          Reference: CLK23  0,1
                          Vector Load: MAUA002        Window Address:   124 = hex   7C    Location:   124 = hex   7C
                          Char offset = -5.900NS      Prod offset = 5.900NS
STRB1 = <1>   111.125NS                                                                   Result =     5.225NS
```

Figure 13. Subroutine ACC: Timing Measurement Output, Debug Mode

The following is an explanation of the debug mode timing measurement output:

Field 1	Test id and the corresponding test number.
Description	This is usually the pinlist name of the pin(s) under test followed by the test type. Test types are as follows: SU = set-up, HO = hold, AS = assertion, NG = negation, TS = tristate, and PW = pulse width.
Verification Pin(s)	The name of the pinlist which has a STROBE assigned to it.
Reference	The name of the pin(s) and the transition which gates or latches the signal which causes the DUT output used for verification, to change.
Vector load	The name of the vector load used for this test.
Window Address	The address at which the verification pins are strobed to determine the pass/fail status at any given iteration of the search.
Location	The location in the vector load at which the transition of the reference signal is to be found.

Char Offset	The amount of offset added to the characterization measurement before the result is compared to specifications.
Prod Offset	The amount of offset added to the timing generator assigned to the pins under test. This value is used in the production test only.
Field 10	This is the name of the scanning timing generator. The equal sign is followed by the scanning timing set in angle brackets, and the value of the timing generator if a pass/fail point is found, or an error message if not.
Result	The result is measured with respect to the point marked *Period* or *Reference Point* in Figures 10 and 11, to allow for both positive and negative measurement values. The characterization offset is then added to it before it is displayed here. This is the same mechanism by which the timing measurement engineering output is derived.

ID	TEST DESCRIPTION			RESULT	28MHZ SPEC	24MHZ SPEC	22MHZ SPEC	20MHZ SPEC	28MHZ MARGIN
F1	SAS	SU	MAX	7.875NS	FAIL> 3.000NS	FAIL> 3.000NS	FAIL> 3.000NS	FAIL> 4.000NS	-4.875NS
F2	SAS	HO	MAX	-5.000NS	3.000NS	3.000NS	3.000NS	4.000NS	8.000NS
F3	CYCLIO	SU	MAX	2.125NS	3.000NS	3.000NS	3.000NS	4.000NS	875.0PS
F4	CYCLIO	HO	MAX	1.125NS	3.000NS	3.000NS	3.000NS	4.000NS	1.875NS
F5	DATA	SU	MAX	6.250NS	18.00NS	21.00NS	23.00NS	25.00NS	11.75NS
F6	DATA	HO	MAX	-1.625NS	0.000S	0.000S	0.000S	0.000S	1.625NS
F7	DRDY0	SU	MAX	2.125NS	3.000NS	3.000NS	3.000NS	4.000NS	875.0PS
F8	DRDY0	HO	MAX	-500.0PS	3.000NS	3.000NS	3.000NS	4.000NS	3.500NS
F9	DATA	SU	MAX	-1.875NS	3.000NS	3.000NS	3.000NS	4.000NS	4.875NS
F10	DATA	HO	MAX	3.250NS	FAIL> 3.000NS	FAIL> 3.000NS	FAIL> 3.000NS	4.000NS	-250.0PS
F13	CS0	SU	MAX	1.375NS	3.000NS	3.000NS	3.000NS	4.000NS	1.625NS
F14	CS0	HO	MAX	1.125NS	FAIL> 0.000S	FAIL> 0.000S	FAIL> 0.000S	FAIL> 0.000S	-1.125NS

AC TEST FAILED 2 OF 12 TESTS

Figure 14. Subroutine ACC: Timing Measurement Engineering Output

The following is an explanation of the timing measurement engineering output, in case it is not intuitively obvious:

ID — The test id.

TEST DESCRIPTION — This is usually the pinlist name of the pin(s) under test followed by the test type. Test types are as follows: SU = set-up, HO = hold, AS = assertion, NG = negation, TS = tristate, and PW = pulse width. The word MAX or MIN means that the specification value(s) are to be interpreted as either maximum or minimum values.

SPEC — There may be several columns of specifications, one for each category. No indicator is given for those specifications that pass, the FAIL> indicator marks failed specifications. The AC test will not fail as a whole unless there are FAIL> indicators in the rightmost SPEC column.

MARGIN — This is the difference between the RESULT and the most stringent specification. A negative margin implies failure of that specification.

```
L THRESHOLD ON ADDR

DATA LOG  PRO MAUB  AT 88/05/14 15:55:21
  DEVICE:  MAUB              SAMPLE: 20        LOT NO:
  DUT I/F: R3                TESTER: 1-1       OPERATOR:
  COMMENT:

TEST   GO/NOGO  DATA      UPPER     LOWER     PIN/VS

90001           1.357V    2.000V    800.0MV   JUDGE

*** BINARY SEARCH ERROR: fails  pass limit( 0.000V  )
L THRESHOLD ON DATA
90002                FAILS MAUA005:   253 =   FD

H THRESHOLD ON ADDR
90003           1.395V    2.000V    800.0MV   JUDGE

H THRESHOLD ON DATA
90004  U-FAIL   2.374V    2.000V    800.0MV   JUDGE


VOH             DATA

90005           4.648V    ........  2.400V    P22
90005           4.648V    ........  2.400V    P23
90005  L-FAIL   -446.0MV   ........  2.400V    P24

VOL             DATA

90006           66.00MV   400.0MV   ........  P22
90006           68.00MV   400.0MV   ........  P23
90006  U-FAIL   5.290V     400.0MV   ........  P24

IOZH            DATA

90007           20.00NA   10.00UA   -10.00UA  P22
90007           20.00NA   10.00UA   -10.00UA  P23
90007           20.00NA   10.00UA   -10.00UA  P24

IOZL            DATA

90008           0.000A    10.00UA   -10.00UA  P22
90008           0.000A    10.00UA   -10.00UA  P23
90008           0.000A    10.00UA   -10.00UA  P24

DYNAMIC IDD

90011           79.20MA   150.0MA   ........  VS1

STATIC IDD

90012           1.600MA   12.00MA   ........  VS1
 DC TEST FAILED
```

Figure 15. Subroutine DCC: DC Measurement Output

The output of the DC test is mostly the standard data logger output of the test system. Only when a pass and fail region can not be identified for the threshold test, does GENTEST output information. The example in the exhibit shows that the test failed at the pass limit which was set at 0V, that is, for a low threshold measurement the input to the DUT which affects outputs at location 253 (hex FD), vector load MAUA005, should be set at logic 0. A

logic 0 was defined as 0V (the pass limit) via IN8, and that should have caused the vector load to pass.

```
DATA LOG  PRO MAUB  AT 88/05/14 15:55:21
  DEVICE:  MAUB            SAMPLE: 20       LOT NO:
  DUT I/F: R3              TESTER: 1-1      OPERATOR:
  COMMENT:


OPEN PINS(CH):    15( 12)   16( 10)   17( 74)   18( 11)   58( 75)
 N PROTECTION DIODE TEST FAILED        0   SHORTS     5   OPENS     0   HIGH IMPEDANCES

OPEN PINS(CH):    15( 12)   16( 10)   17( 74)   18( 11)   58( 75)
 P PROTECTION DIODE TEST FAILED        0   SHORTS     5   OPENS     0   HIGH IMPEDANCES
```

Figure 16. Subroutine DIODE: Diode Test Output

The pin numbers represent logical or software designations, whereas the channel numbers in parentheses, refer to hardware.

```
PROTECTION DIODE TEST PASSED

10001 MAUTPERP pass          10002 MAUADDF  pass          10003 MAUATAN1 pass          10004 MAUCMP1  pass
10005 MAUCOS1  pass          10006 MAUCOS2  pass          10007 MAUCRND1 pass          10008 MAUD001A pass
10009 MAUD002A pass          10010 MAUD003A pass          10011 MAUD004A pass          10012 MAUD004B pass
10013 MAUD005A pass          10014 MAUD006A pass          10015 MAUD007A pass          10016 MAUD007B pass
10017 MAUD008A pass          10018 MAUD009A pass          10019 MAUD010A pass          10020 MAUD010B pass
10021 MAUD010C pass          10022 MAUD010D fail at 1594  10023 MAUD010E pass          10024 MAUD010F pass
10025 MAUD010G pass          10026 MAUD010H pass          10027 MAUD010I pass          10028 MAUD010J pass
10029 MAUD010K pass          10030 MAUD010L pass          10031 MAUD011A pass          10032 MAUD011B pass
10033 MAUD012A pass          10034 MAUD013A pass          10035 MAUD013B pass

FUNCTIONAL * FAIL    1 OF  35
```

Figure 17. Subroutine FNC: Functional Test Output

```
LIMITING OPERATING FREQUENCY
20001  MAUTPERP =   37.3MHZ
20002  MAUADDF  =   38.7MHZ
20003  MAUATAN1 =   31.7MHZ
20004  MAUCMP1  =   39.1MHZ
20005  MAUCOS1  =   31.7MHZ
20006  MAUCOS2  =   37.3MHZ
20007  MAUCRND1 =   36.1MHZ
20008  MAUD001A =   39.9MHZ
20009  MAUD002A =   39.1MHZ
20010  MAUD003A =   37.7MHZ
20011  MAUD004A =   38.7MHZ
20012  MAUD004B =   38.7MHZ
20013  MAUD005A =   38.7MHZ
20014  MAUD006A =   39.1MHZ
20015  MAUD007A =   39.1MHZ
20016  MAUD007B =   39.1MHZ
20017  MAUD008A =   35.1MHZ
20018  MAUD009A =   36.6MHZ
20019  MAUD010A =   39.1MHZ
20020  MAUD010B =   39.1MHZ
20021  MAUD010C =   39.1MHZ
20022  MAUD010D =   39.1MHZ
20023  MAUD010E =   39.1MHZ
20024  MAUD010F =   39.1MHZ
20025  MAUD010G =   39.1MHZ
20026  MAUD010H =   39.1MHZ
20027  MAUD010I =   39.1MHZ
20028  MAUD010J =   39.1MHZ
20029  MAUD010K =   39.1MHZ
20030  MAUD010L =   39.1MHZ
20031  MAUD011A =   38.1MHZ
20032  MAUD011B =   36.7MHZ
20033  MAUD012A =   39.1MHZ
20034  MAUD013A =   38.7MHZ
20035  MAUD013B =   38.1MHZ

MAXIMUM FREQUENCY:
20008  MAUD001A =   39.9MHZ

MINIMUM FREQUENCY:
20005  MAUCOS1  =   31.7MHZ
```

Figure 18. Subroutines FQBC or FQLC: Frequency Test Output

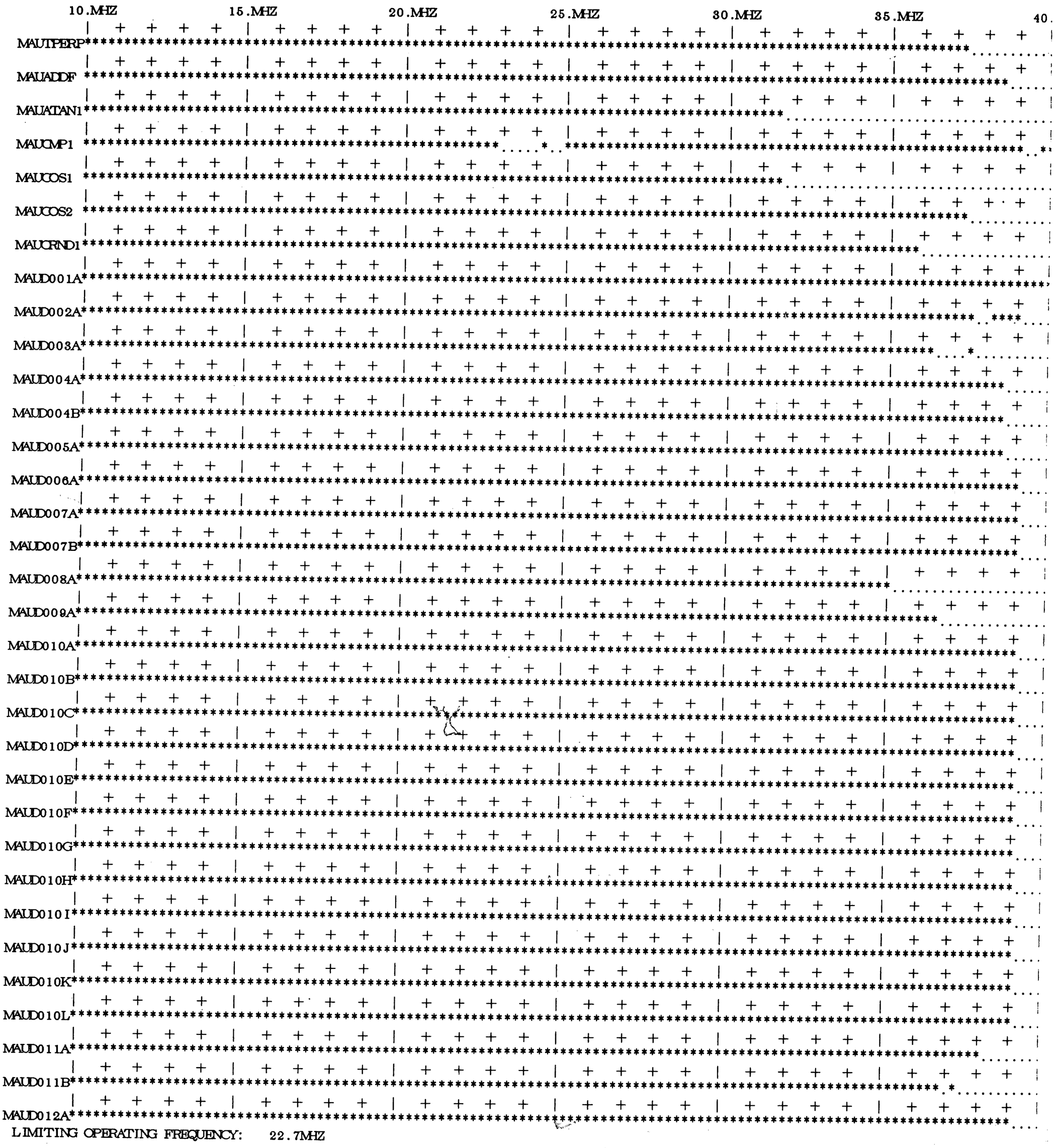

Figure 19. Subroutine FQLC: Linear Search Frequency Test Output

For the frequency histogram, the asterisk is the PASS indicator and the period is the FAIL indicator. Increments are at 250KHz.

```
STATION 1    MAUB2 PACKAGE TEST  88/05/14 15:48:54
PACKAGE NUMBER MAUB2-1

 DIODE           PASS
 FUNCTIONAL      PASS
 FREQUENCY    *  FAIL     1 OF 120 AT    20MHZ
 AC              PASS      28MHZ
 DC              PASS


BAD PART
```

Figure 20. Compact Characterization Test Output

8. VITA

DATE OF BIRTH:	August 3, 1947	
PLACE OF BIRTH:	Landau/Pfalz, West Germany	
CITIZENSHIP:	USA	
MILITARY SERVICE:	November 1966 - August 1968	US Army MOS: Radio Operator National Defense Service Medal Vietnam Campaign Medal Vietnam Service Medal Army Commendation Medal

EDUCATION:

May 1979	Diploma Electronics Technician DeVry Technical Institute, Woodbridge, NJ
October 1985	Bachelors Degree Electrical Engineering/Computer Science Fairleigh Dickinson University, Teaneck, NJ

RELEVANT EXPERIENCE:

June 79 - February 81	AT&T Bell Laboratories: device processing
February 81 - present	AT&T Bell Laboratories: device testing