Architectural verification of advanced storage controllers

by S. S. Soult

The architectural complexity of advanced storage controllers has increased to a point at which architectural verification methods based on document inspections and reviews are no longer effective. To facilitate the architectural verification process, a high-performance simulator (TurboSim) has been developed for architectural-level verification. The TurboSim application includes a set of architectural-level models, which represent essential architectural components, and an automatic test case generator (ATG). The TurboSim ATG is used to generate realistic representations of customer direct access storage device (DASD) track data. The track data are used to drive different TurboSim simulation scenarios. After demonstrating its effectiveness as an architectural verification tool, TurboSim was enhanced to support the automatic generation of hardware test cases. These hardware test cases are used to ensure that the hardware implementation matches architectural specifications.

Introduction: The architectural development and verification process

An essential aspect of the advanced storage controller architectural development and verification process is architectural verification. The overall process is shown schematically in **Figure 1**. The individual steps required are described in the following subsections.

- *Initial draft*

The development of an advanced storage controller subsystem architecture is initiated by a small group of developers that has been organized as an architectural team. The team develops an initial draft of the architecture under the direct supervision of a chief architect, who is responsible for the overall design and implementation of the architecture. He is responsible for ensuring that the members of the architectural team convert the architectural concepts into prose descriptions that are clear, concise, and accurate.

The chief architect must ensure that the architecture is unified and possesses conceptual integrity. The importance of maintaining the conceptual integrity of an architecture is discussed by Brooks [1], who states that "Conceptual integrity in turn dictates that the design must proceed from one mind, or from a very small number of agreeing resonant minds."

- *Architectural revision*

The initial draft of the architecture is taken through several iterations by the architectural team as it is refined. The initial development of an advanced storage controller subsystem architecture can take several months to complete.

0018-8646/96/$5.00 © 1996 IBM

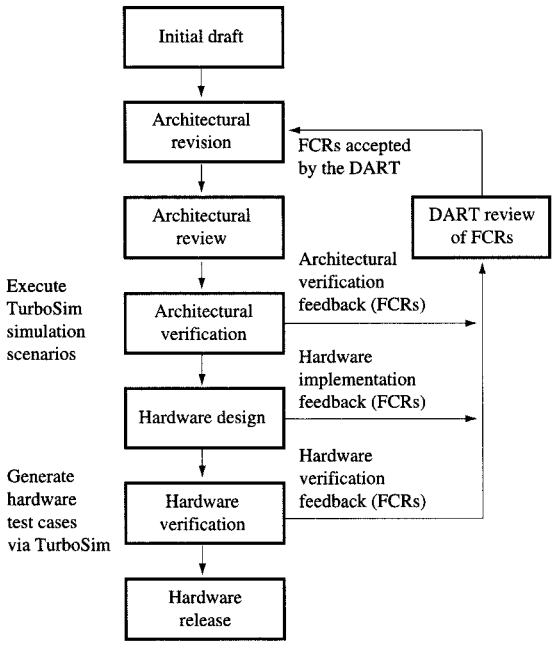

Figure 1

Architectural development and verification process.

- *Architectural review*

After the architectural team completes the initial draft of the architecture, it is made available for general review. A formal document review is used to uncover problems in the initial draft, and these are resolved by the architectural team during the development of the final draft of the architecture.

After the final draft is completed, a formal acceptance review is conducted, and any remaining problems are addressed before the first edition of the architecture is published. The document is placed under formal change control after the first edition is published.

- *Formal change requests*

After the architecture is placed under formal change control, a Formal Change Request (FCR) process is used for all functional changes to the architecture. Advantages of the FCR process include the following:

1. The FCR process tends to eliminate misunderstandings and misinterpretations of the architecture by people outside the core architectural development team. This reduces the number of problems written against the architecture.
2. FCRs are placed on a system conference disk that is easily accessible by all advanced storage controller project developers. This ensures that they are kept informed of proposed architectural changes.
3. The FCR process provides an opportunity for qualified commentators other than the members of the architectural team to participate in the development, review, and enhancement of the architecture.
4. The FCR process also provides a means for processing the FCRs by the Design and Review Team (DART) in a timely manner. The DART is a small independent group of individuals who need not be members of the architectural team.

- *Architectural verification*

After the initial draft of the architecture is completed by the architectural team, the architecture is verified. Because of the tremendous complexity of recent advanced storage controller subsystem architectures, traditional document inspection and review methods are inadequate for architectural verification. For example, some of the subsystem data transformations are so complex that it is no longer practical to check them by hand.

One approach to solving the problem of checking very complex architectures is to build a set of software models of the architecture and then execute them. The principal purpose behind the development of the TurboSim application was to provide a simulation environment for models written at the architectural level.

A similar approach had been used previously for the development of the Systems Network Architecture (SNA). Smith and West[1] found that the direct implementation of an architecture solves two important problems [2]:

1. It determines what constitutes an implementation of the architecture.
2. It can be used to test whether a particular implementation complies with the architecture.

- *Hardware design and implementation*

As essential elements of the architecture are verified, the hardware design and implementation can proceed. With careful planning and prioritization, the verification, design, and implementation can proceed as parallel activities, rather than being done sequentially.

When problems are found with the architecture, FCRs are written and presented to the DART. FCRs that are accepted by the DART are incorporated into the architecture and then reviewed and verified as part of the TurboSim modeling effort.

[1] One of the principal differences between the approach taken by Smith and West and ours is that they used a special language, the Format And Protocol Language (FAPL), to write their models. Rather than using a specialized modeling language, we decided to use a modern Object-Oriented Programming (OOP) language, C++, to write our models. One of the advantages of using C++ as a modeling language is that it provides opportunities for code reuse during the development of product microcode.

• *Hardware verification*

To ensure that the hardware implementation is correct, a means must be provided to compare the results of architectural simulation with the results produced by the hardware. We accomplished this by enhancing the TurboSim models to support the automatic generation of Subsystem Test case Language (STL) hardware test cases. By using the architectural-level models for hardware test case generation, we discovered not only hardware implementation problems but some architectural problems as well. The types of architectural problems found during this phase tended to be related to ambiguous, incomplete, or missing parts of the architecture.

Overview of TurboSim

• *Characteristics*

TurboSim is a high-performance simulator designed primarily for architectural verification. There are two different versions of TurboSim, which run under different operating systems and on different hardware platforms:

- TurboSim/DOS runs under DOS on an IBM Personal Computer (PC); it is used primarily for code development, architectural verification, and education and training purposes.
- TurboSim/AIX runs under AIX® on an IBM RS/6000™ server. TurboSim/AIX is used for essentially the same purposes as TurboSim/DOS, but in addition it is used by the hardware test case writers for STL test case generation. The STL test cases have been used primarily for hardware verification. However, they have also been used in a limited way for software verification.

The TurboSim code is written in the C++ programming language, which is an excellent language for constructing simulation models. During the development of C++, Bjarne Stroustrup relied heavily on his prior experience in building a simulator while he was at Cambridge [3]. Therefore, the C++ programming language has many attributes and features that make it an excellent choice for building the TurboSim models.

The data structures described in the architecture have been constructed as accurately as possible, and have been implemented as C++ classes[2] in order to take full advantage of Object-Oriented Programming (OOP) techniques.

A simple user interface technology enables the TurboSim application to be ported across different operating systems. The different TurboSim ports use the same User Interface (UI) technology and have the same operational characteristics. The TurboSim user interface conforms to the entry model guidelines specified in [5].

Customer data are represented by data patterns contained in TurboSim track data files (TDFs). Each track of data contained within a TDF is considered to be a test case. TDFs are normally generated within TurboSim. If desired, they can be edited with a standard text editor. The ability to generate TDFs automatically and then use them to run simulation scenarios represents a very powerful Automatic Test case Generation (ATG) capability.

The TurboSim application supports both batch and interactive modes of operation. The default mode of operation is the interactive mode. To run the application in batch mode, a batch flag command-line argument is specified by the user. The TurboSim application command line arguments are as follows:

- The *Flag* command-line argument is optional and is used to specify the program mode of operation. The various modes of operations are batch (background), debug, and interactive (foreground).
- The *TrkFile* argument specifies the name of the TDF to be used. This file must be in a specific format to be recognized by the program. The *TrkFile* argument must be specified if a *LogFile* argument is to be specified.
- The *LogFile* argument specifies the name of the simulation log file to be used. If the *LogFile* argument is not specified, the name of the simulation log file will be constructed from the TDF name.

The *TrkFile* and *LogFile* arguments are normally specified for batch jobs. This permits the user to use different file names for batch jobs which are run consecutively. However, the *TrkFile* and *LogFile* arguments can be specified for the other modes of operation. The arguments then specify the default names used in the menus for the TDF and simulation log files.

Most of the TurboSim menus and displays have context-sensitive help, including some of the information contained in the TurboSim User's Guide, which can be either be viewed on-line or printed.

TurboSim features

The TurboSim primary menu lists options in the sequence that a user would normally follow when using the TurboSim application:

1. Build a track data file.
2. Run a specific simulation scenario.
3. Examine the results of the simulation scenario.
4. If desired, build STL test cases.

[2] A C++ "class" is a language construct which allows the programmer to encapsulate data members and the member functions (or methods) that operate on the data. A good introduction to C++ classes can be found in [4].

These features are more completely described in the following subsections.

- *Build track data file*

The "build track data file" feature is used to build a TDF. The TDF contains the track information normally transferred to the subsystem by the system via the channel interface. The TDF information is required to run a TurboSim simulation scenario. Since TDFs must conform to a specific format in order to be recognized by the simulator, users should always use the TurboSim build track data file feature to construct an initial version of a TDF. After a TDF has been constructed, it can be modified with an ordinary text editor.

To stress certain boundary conditions within the architecture, it is essential to be able to build specialized TDFs by hand. Handcrafted TDFs have been very effective in uncovering architectural problems. This finding is in accordance with work done by Cross et al. [6], who concluded that random testing by itself was not adequate for exposing errors. However, when it was combined with boundary and special value (error statistics by volume, or ESV) testing, it proved to be a very effective method.

Build track data file options

The build track data file options permit the user to build either a regular or irregular Count Data (CD) or Count Key Data (CKD) TDF. The regular CD or CKD TDFs contain fields of the same length for all of the records on the track. The irregular CD or CKD TDFs contain fields, which differ in length, for each record on the track.

After selecting one of the build track data file options, the user is prompted for a TDF file name. If a file name is not specified, the "sim.dat" default file name is used.

The user is then prompted for a random data-generation decision. If the user wants random TDF data to be generated, a "y" is typed after the prompt. With random data generation, the content of each TDF is unique.

If the user does not want a random TDF data pattern to be used, but a fixed data pattern instead, an "n" is typed after the prompt. With a fixed data pattern, each TDF will be the same (except for the number of tracks).

The last thing that the user is prompted for is the number of tracks. For TurboSim/DOS it is recommended that a small number (1–10) of tracks be chosen for simulation if the user does not use a hard-disk caching program. The reason for this is that the TurboSim application writes and reads several intermediate files during the course of a simulation scenario. Therefore, total simulation time is related to the total number of tracks processed.

Track data generation algorithm

The algorithm for generating the track information is based upon the information contained in [7]. The space in cells consumed by a record can be calculated via the following equations:

$$Space = C + K + D,$$

where $C = 10$, K depends upon the Key Length (KL), and D depends upon the Data Length (DL). If $KL = 0$, $K = 0$. If KL is not equal to 0,

$$K = 9 + \frac{KL + 6KN + 6}{34},$$

where $KN = (KL + 6)/232$. The number of cells used by the data field is

$$D = 9 + \frac{DL + 6DN + 6}{34},$$

where $DN = (DL + 6)/232$. Both K and D are rounded up to the nearest integer.

- *Run simulation scenario*

A simulation scenario menu is provided to allow the user to choose a specific simulation scenario for execution. Several different simulation scenarios can be selected. After the user has selected a simulation scenario, the TDF and simulation log names for the scenario can be specified. If the file names are not specified, the default file name for the TDF will be "sim.dat" and the default file name for the simulation log will be "sim.log".

After the input and output file names have been selected, the user can either accept or modify the standard simulation scenario defaults. The specified simulation scenario is then run. **Figure 2** shows the data transformations performed in a typical simulation scenario.

The TDF information is processed at the beginning of a simulation scenario. It is used to build an internal representation of the track data structure that is used by node model 1.

The standard data transformation scenario consists of two basic operations:

1. A format write operation is performed. During the format write operation, a single track of data is written to the direct access storage device (DASD).
2. A read track operation is performed. During the read track operation, a single track of data is read from the DASD. The standard data transformation scenario is successful if the data received by node model 1 during the read track operation are the same as the original track data.

A total of ten transformations are performed on the data as they move from node 1 to node 2 to node 3, to the DASD, then back to node 3, back to node 2, and finally back to node 1. The data perform a round trip as they pass through transformation to transformation in a counterclockwise direction. A total of ten different transformations are performed on the data during the round trip through the subsystem. If an error occurs during the round trip, simulation is stopped, and an error message is presented identifying the source of the error.

The results of the simulation session are written to the simulation log file specified during session initialization. After the simulation log file has been written, a "browse" function is invoked automatically by the simulator to browse the contents of the simulation log file.

- *Browse simulation file*

The TurboSim browse simulation file feature can be used to browse a simulation file created during the execution of a simulation scenario. This feature is normally used to view simulation logs. However, it can also be used to view other text files, such as the TurboSim TDF files. These are standard text files which can be browsed with no special processing.

- *Model displays*

The model display options are used to examine the internal model data structures constructed during a simulation scenario. The data structures which can be displayed depend upon the specific simulation scenario selected.

The contents of the various data structures are preserved after a simulation session, so that they can be examined. The data structures can be viewed by selecting one of the options provided on a model display menu.

Since the data structures are preserved after a simulation session, the space that they consume is not freed. However, the space would be freed automatically at the beginning of a new simulation session if simulation were to be run again.

- *Data stream displays*

This option is used to examine the data streams passed between the nodes via the node connections. As in the case of the model displays, the data streams which can be displayed are dependent upon the specific simulation scenario selected.

- *Save model data*

This option is used to save the model data in a file for future reference. In many cases it is important to be able to analyze control-block information after a simulation session has been completed.

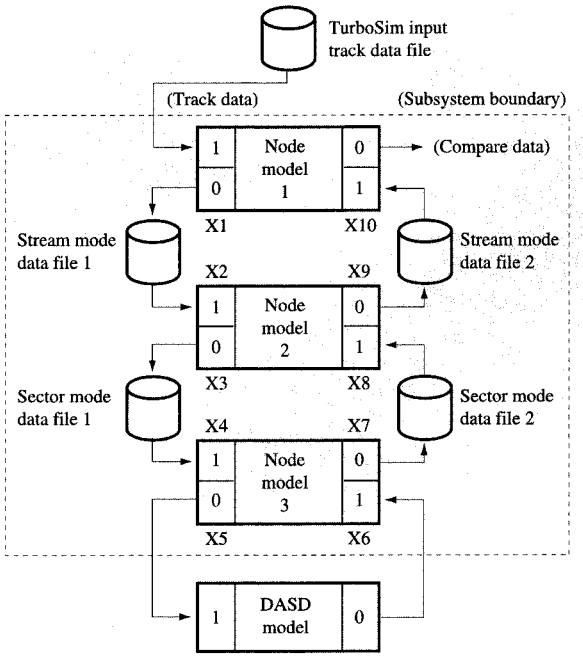

Figure 2

Standard data transformation scenario.

The model data are saved in the form of STL files, which can be used later to construct hardware verification test cases. In addition, individual STL macros can be generated from many of the TurboSim control-block displays. When the user presses a function key from one of the model control-block displays, an STL macro for the currently displayed control block is generated and placed in a user-specified file.

The control-block addresses generated by TurboSim/AIX are unique to the IBM RS/6000 environment. Therefore, they must be replaced by symbolic addresses, which can be converted to real addresses by the STL translator. Because the STL translator creates an address/data map which can be loaded into memory on the target hardware, it is commonly referred to as the "mapper."

Architectural verification

The principal focus of our architectural verification was to ensure that the data transformations shown in Figure 2 (X1 through X10) were performed correctly. Although specific numbers were not tracked, it is possible to make some informal calculations related to the comprehensiveness of the architectural verification.

2,000 Average number of test cases per release per programmer per scenario
×3 Number of principal scenarios run

6,000 Average number of test cases per release per programmer
×3 Average number of TurboSim programmers

18,000 Average number of test cases per release
×10 Number of TurboSim releases

180,000 Average number of test cases
×10 Average number of data transformations per test case

1,800,000 Total number of data transformations

Figure 3

Data transformation calculations.

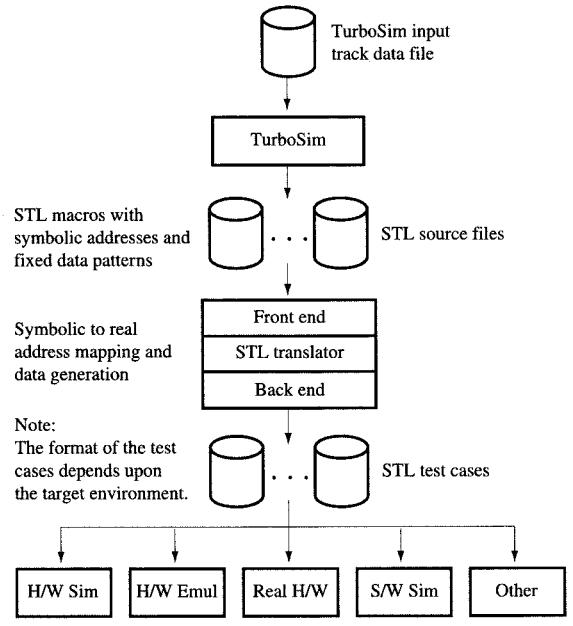

Figure 4

Overview of hardware verification support.

Most interesting is the number of data transformations performed during the architectural verification process.

In the calculations shown in **Figure 3**, a test case is considered to be the data for a single track contained within a TDF. Test cases are typically run in batches of 500 each during TurboSim regression testing. Regression testing is an essential aspect of the TurboSim release process.

These numbers give a sense of how comprehensive the architectural verification was, using random test case generation. In addition, the hardware test case writers built TDFs by hand which stressed various boundary conditions. These handcrafted test cases were very effective in discovering boundary condition problems, both within the architecture and in the hardware implementation.

Despite the fact that they are very coarse, the numbers shown in Figure 3 are useful from an order-of-magnitude perspective. They also demonstrate that the data transformation part of the architecture was verified very thoroughly.

Hardware verification

TurboSim provides support for hardware verification by generating STL files. The process shown in **Figure 4** is used for hardware verification.

- *Hardware verification process*

The hardware verification process is described in more detail as follows:

1. The first step in the process is to build a TDF. To stress important boundary conditions, the hardware test case writers build handcrafted TDFs with the desired track characteristics.
2. Next, a specific TurboSim simulation scenario is run. The data structure and data file information generated during the simulation scenario are used to generate the STL source files.
3. Typically, a hardware test case writer then merges several of the STL source files together and adds additional test case header information. This enables the mapper to build an STL test case suitable for loading into the target environment.

   In addition, a hardware test case writer can change the content of an STL source file for error-injection purposes. The ability to alter any bit within any field of a data structure represents a very flexible and powerful Fault INSertion (FINS) capability. FINS support is essential to ensure that the hardware will handle error conditions properly.
4. The mapper is used to translate STL source files into binary test case files that can be loaded into the target environment. The mapper consists of a front end, which is responsible for parsing the source files, and a back end, which is responsible for building a binary test case file for a specific target environment. Typically, there are different mapper back ends for each target environment. This is because different binary test case file formats are normally required for different environments.
5. After the binary test case files have been generated, they are loaded into the target environment. Normally,

two different versions of the data structures are used by a test case. Data structures containing initial values are provided to place the hardware in an initial state. Data structures containing final values are provided so that a comparison can be made between the expected results and the actual results produced by the hardware.

At present, the only paths which have been completed and used are the path to the hardware simulation environment and the path to the software simulation environment.

- *STL test case production*

Figure 5 shows the number of STL test cases written using TurboSim during a 16-week period. Of the 322 STL test cases written during that period, 284 ran successfully on simulated hardware and were marked complete. This exceeded the planned value of 220 STL test cases by a comfortable margin. These numbers show that the use of TurboSim STL ATG capabilities permitted the hardware test case writers to exceed their plan.

However, what we found most interesting about the introduction of the new STL ATG technology is how the hardware test case writers utilized the extra time saved through automation. Rather than using their extra time to write test cases, as we had expected, they used it to help the hardware designers debug the hardware. This phenomenon is shown by the test cases "held by H/W" curve in Figure 5.

In week 6, the number of test cases delayed by hardware problems had reached 60. This situation represented a tremendous crisis for the hardware organization, because only one test case had passed at this point in time. However, at this time the new TurboSim STL ATG capabilities had reached the point where the hardware test case writers could save a significant amount of time in producing their weekly quota of test cases. They used the extra time saved to contribute to the hardware debug effort.

By the next week, the results were already evident. The number of test cases delayed by hardware problems was reduced to 41, and the number of test cases passed increased to 26. As incremental improvements were made in the TurboSim STL ATG capabilities, the hardware test case writers spent less time writing test cases and more of their time in the hardware debug effort. This is evidenced by the fact that the number of test cases delayed by hardware continued to decrease from week 7 to week 14 and then remained constant at nine test cases, a very low number.

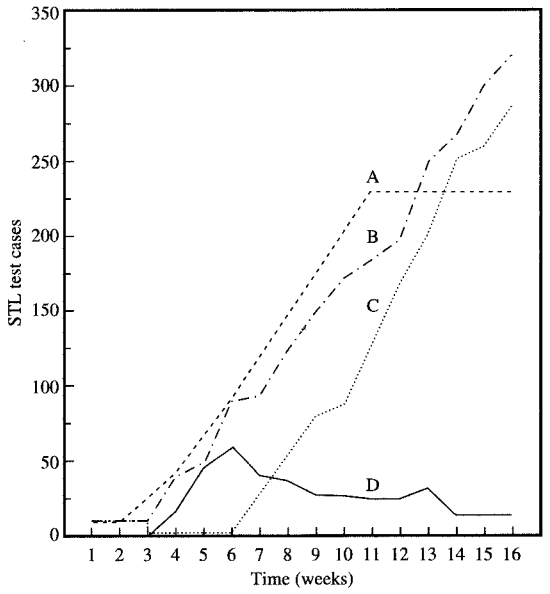

Figure 5

Number of STL test cases generated in a 16-week period using TurboSim: Curve A, plan; Curve B, written; Curve C, complete; Curve D, held by H/W.

Conclusions

Simulation has proved to be an excellent technique for uncovering architectural-level problems before they are propagated into the design. At the time this paper was written, 24 significant problems with the architecture had been discovered by the TurboSim team. These problems were not discovered through use of traditional techniques, such as detailed document reviews and inspections.

Two architectural problems were discovered when the TurboSim models were used to build hardware verification test cases. Since the architectural verification process is an ongoing activity, it is anticipated that a small number of problems will continue to be found. However, the ongoing rate of problem discovery has been far less than in the initial architectural verification stages.

A summary of the problems found with TurboSim is provided below:

Problem type:
- Architectural = 12
- Documentation = 12

Problem disposition:
- Resolved = 18
- Dropped = 5
- Pending = 1

In addition, we determined that it was practical to use architectural-level models for generating hardware test

cases. When an ATG capability for constructing hardware test cases is available, the hardware organization can spend more time testing and debugging the hardware. Removing both architectural-level and hardware design problems very early in the design cycle has significantly reduced the advanced storage controller subsystem development cost, improved the product development schedule, and will result in a much higher-quality product being delivered to our customers.

Acknowledgments

The author would like to thank James Brady for his support and assistance throughout the duration of the TurboSim project. Without his foresight and vision, the TurboSim project would never have been launched. In addition, without his assistance the construction of the architectural-level models would not have been possible. The author would also like to acknowledge the contributions made by the members of the TurboSim team. The principal contributors to the TurboSim project were Peter Baiko, Ken Brown, Kevin Bui, Anthony Cascella, Larry Garibay, Gene Hopp, Charlotte Hsieh, Al Rogers, Calvin Tang, and Mai Tran.

References

1. F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley Publishing Co., Reading, MA, 1982, p. 44.
2. F. D. Smith and C. H. West, "Technologies for Network Architecture and Implementation," *IBM J. Res. Develop.* **27,** 68–78 (1983).
3. B. Stroustrup, "A History of C++: 1979–1991," *ACM SIGPLAN Notices* **28,** No. 3, 272 (March 1993).
4. S. B. Lippman, *C++ Primer 2nd Edition*, Addison-Wesley Publishing Co., Reading, MA, 1991, pp. 215–221.
5. *Systems Application Architecture Common User Access Basic Interface Design Guide*, SC26-4583-0, First Edition (December 1989), Chapter 4.
6. J. H. Cross, K. Chang, W. H. Carlisle, and D. B. Brown, "Expert System Assisted Test Data Generation for Software Branch Coverage," *Data & Knowledge Eng.* **6,** 281 (1991).
7. *IBM 3390 Direct Access Storage Reference Summary*, GX26-4577-03, Fourth Edition (September 1991), p. 3.

Received March 10, 1995; accepted for publication August 9, 1996

Steven S. Soult *Abbott Laboratories, Abbott Critical Care Systems Division, 1212 Terra Bella Avenue, Mountain View, California 94043 (ssoult@abbotthpd.com).* At IBM, Mr. Soult was the technical team leader of the TurboSim development team in San Jose. His research at IBM involved the development of high-performance simulation environments for architectural, hardware, and microcode verification. His interests also included the management of high-performance technical teams. He received an IBM Outstanding Technical Achievement Award for his work on the TurboSim project. Mr. Soult received a B.S. in electrical engineering from Santa Clara University in 1971. After serving four years as an officer in the United States Navy, he returned to graduate school and received an M.S. in electrical engineering and computer science from the University of California at Berkeley in 1978. Later, while working full-time at IBM, he received an M.B.A. degree in management from Santa Clara University in 1988. He is a member of the IEEE Computer Society.