AN EMULATION METHODOLOGY FOR THE DEVELOPMENT OF ELECTRONIC SYSTEMS FOR TRAINING

Roy Latham

Scientific Staff
The Singer Company, Link Flight Simulation Division
Sunnyvale, California 94088-3484

ABSTRACT

The traditional method of developing electronic systems, especially complex low-production systems, has been to do a careful design and then build and refine a hardware prototype. But beyond a certain level of complexity, a level which, for example, digital generators for flight simulation have clearly passed, this method of hardware refinement is not practical, and a development methodology that leads to a much greater level of design certainty should be adopted. The method proposed comprises high level language software emulation of algorithms, gate level verification of logic designs, a staged system build up that couples emulations with real hardware, and a comprehensive system of diagnostics. The diagnostic system supports both hardware design verification and system maintenance.

A DESIGN PROBLEM

Every designer of electronic systems works within constraints of schedule, budget, and product performance. For each product there will be particular aspects of the product performance that prove especially challenging, and the methods of designing the product are driven accordingly. There is a class of electronic systems for training where the predominate concerns are the complexity of the system and the tight schedules imposed for completion of the designs. This paper deals with a software emulation methodology that holds promise of improving our ability to cope with this problem.

The driving force that makes some training systems complex is the need to recreate an inherently rich and complicated environment to present a usefully complex situation to a trainee. For example, a complex acoustical environment must be created to train a sonar operator, or detailed radar imagery must be created and continually updated for aircraft crew training. The case examined here is the construction of systems for the generation of out-the-window scene imagery for crew training in such detail-intensive tasks as low altitude flight in helicopters.

For digital image generation, the most complex electronics systems presently practical are still not adequate to produce a visual scene indistinguishable from the real world. One limit to system complexity is the reliability of large numbers of electronic components, about one hundred thousand integrated circuits being the practical limit in systems of the recent past. However, with VLSI technology that is available now, the real limitation is the ability to design systems of great complexity within schedule constraints.

The reason that schedule constraints are so important to training system design is that the procurement of training systems follows the procurement of the actual system for which the training system is sought. But ideally, the training system should be made available before the real system. The situation would be impossible were there not a great deal of commonality from one system to the next in the systems of high complexity. Nonetheless, the overall effect is a design intensive process; production runs are rarely large enough to significantly amortize the cost of design, and design costs are limited by the schedule.

This means that the key to the important problems of complexity, schedule, and cost is the efficiency of design. Consequently, the methods used in designing complex systems are of considerable interest.

THE ROLE OF DESIGN METHODOLOGY

Simple electronic circuits can be designed satisfactorily with informal methods: sketch the circuit, build it up, and test it. If the circuit is not very complicated, it may work exactly as intended or just take a few hours to debug. For more complex systems, more formal design methodologies come into play. At some level of complexity, there will be a hierarchy of design, with a top-level system design, individual subsystem specifications, circuit card specifications, and circuit card detailed designs. We are forced to hierarchical design methodologies by the mind's limited capacity for the comprehension of detail; as systems become more complex we are forced to higher and higher levels of abstraction.

How, then, can we be sure that the concepts envisioned at the highest level are implemented accurately in the detailed level of design? It is more than just correctly implementing a circuit block diagram. The most abstract level of system specification is now usually in the form of mathematical algorithms. There are two major contending schools of thought as to how bring an electronic system implementation into correspondence with its system conception. I shall call these two approaches "hardware prototyping" and "software prototyping".

The hardware prototyping school of thought has evolved directly from *informal design methodologies and* anticipates extensive debug of prototype hardware. The key to timely system development is seen as the very rapid construction of a prototype system using flexible construction techniques (like wire-wrapping) so that subsystem and system level problems can be met and conquered from as early a point as possible in the design cycle. Project success hinges upon a very few key individuals who can quickly produce a system design that is nearly correct, and upon a lengthy debug cycle. The essence of the hardware prototyping approach, especially the debugging effort of heroic proportions, was captured in a popular book a few years ago [1].

The software prototyping school of thought has emerged in part from the advance of integrated circuits technology, where the cost of design errors in remaking integrated circuits is so high that alternatives are essential. The concept is to exercise a software model of the design prior to the construction of the device. Originally, the concept was restricted to the verification of a circuit design, something expressable in gates rather than mathematical algorithms. But subsequently the tools have evolved into forms more suitable for system level application.

Another origin of software prototyping concepts on a broad systems' level was the space program, perhaps the most clear cut example of a requirement that could not be met entirely by hardware prototyping. But applications like the space program and VLSI design were treated with specialized tools that were rarely applicable to other contexts. Only in the past three or four years have software prototyping methodologies appeared as a possible alternative to hardware prototyping in general applications.

Because software prototyping is done in a computer, its use presupposes an array of programs and expertise not traditionally associated with hardware design, and this circumstance poses many problems. For example, who would wish to be the first to exercise, and presumably debug, a new methodology with a new set of tools? On a complex and time critical project?

Before addressing the problems of software prototyping, and making further comparisons with hardware prototyping, it is best to first describe in more detail what is involved in a software emulation methodology.

A SOFTWARE PROTOTYPING METHODOLOGY

Overview

There are four basic steps in the software prototyping methodology proposed here. These steps are outlined as follows.

1) **Emulation of algorithms** is done by writing programs on a general purpose computer in a high level language like Fortran. This is done to verify that the algorithms are designed correctly and to determine the arithmetic accuracy required for hardware implementation.

2) Circuit specification, design, and verification is performed on a computer-aided design system. System specifications derived from the high level language emulations are used to prepare gate-level circuit designs. Schematics are entered on the design system, and the logic and timing verifications are performed.

3) Circuit construction and test are performed for each card independently, with test performed on a micro-computer driven test station. Test data derived in the previous design phases is used to find errors in manufacturing, and other design problems, if any.

4) System build-up and test uses the individually tested cards in the system packaging. Data from the high level language emulations is used to test the cards in functional groups. Cards are added until the system is complete.

The design methodologies being developed for VLSI circuit design concentrate heavily upon the second phase of the system development methodology. Only recently has the level of integration become large enough to permit system-level complexity on a single chip, and this is still rare. Consequently, questions related to algorithmic design and system build-up could be previously considered separately from chip design. With the advent of user-designed integrated circuits that encompass system-level complexity, design methodology issues will be forced upon users as well as chip design specialists. The system development methodology described in this paper is a step towards entry into that arena of development.

Another characteristic of emulation methodologies is that data interfaces abound. We expect high level language software to drive functional groups of circuit cards, and we expect test vectors used in logic verification software to be applied to actual cards. Design data must be transfered to circuit card and backplane manufacturing processes. These interfaces must be considered from the earliest stages of system development if the development methodology is to be used successfully.

High-Level-Language Emulations

The purpose of high-level-language emulation is to prove the key concepts of a system before building it. In the early stages this means programming and testing the equations and algorithms to be embodied in the system hardware. Later, the emulations are used to verify the accuracy requirements, processing speed requirements, buffer memory sizes and other implementation sensitive considerations.

The main sense of "verification" in these contexts is that the system produces results that are consistent with the expectations of the design. The design expectations may be quantifiable. For example, system specification may directly establish that a quantity must be computed with total error less than a specified value. With complex training systems, however, the specifications often have important requirements that are not directly quantified. For example, a visual scene for flight training may be required to have "no distracting visual anomalies", i.e. no distracting artifacts of digital construction like stair-stepped edges on objects depicted in the scene. Even less specifically, the system may be required to be "adequate for any training purpose of the simulation" or words to that effect.

There may be preconceptions of the quantitative requirements needed to meet non-specific objectives, but it is often not possible to verify the assumptions without the subjective opinions of human subjects. After all, the ultimate test of whether or not an effect is distracting is to show it to people and ask them if they find it distracting.

Consequently, one important implication of emulation development is the need for specialized hardware for output of the emulation results. For visual systems a "frame buffer" can be used to store the picture elements of an image as they are computed, and to read out the picture elements and convert them for display on a monitor. Video disc or tape can then be used to build animated sequences from individual frames that each require many minutes to compute by emulation. Moving sequences show temporal effects and serve to run through many different cases of data and parameters.

In a practical work environment, users of the emulation environment work with computer terminals through a data switch with any one of several computers. Frame buffer, video disc, and video tape outputs are converted to standard television formats and routed back to the user offices over a sort of cable television system (Fig. 1). This allows users to develop and test emulation software in an office setting, (Fig. 2) while sharing the use of expensive video peripherals. The limitations of the standard television format require critical evaluations of color or edge sharpness to be done with direct connection to calibrated laboratory displays, however.

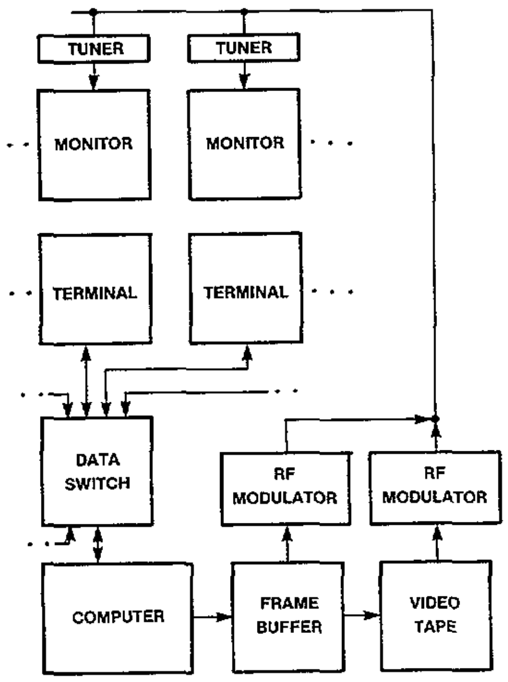

Figure 1. Elements of a laboratory for the emulation of real-time image generation systems.

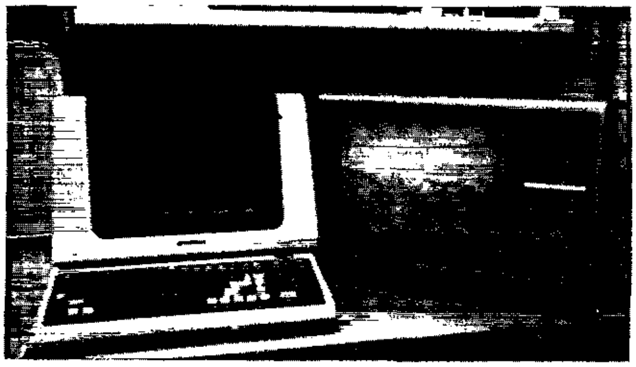

Figure 2. Office workstation with computer terminal and monitor.

Even with the best equipment, emulations cannot substitute for analytical work and careful engineering. An obvious limitation is the non-interactive nature of the presentations; computer emulations run so much slower than actual training system hardware that true training system interaction is ruled out. An additional consequence of lengthy computation times is that emulation sequences must be carefully planned to try to encompass all of the worst case conditions of data and parameters. A separate engineering analysis is required to indicate the circumstances likely to show problems. Finally, the emulation sequences must be critically evaluated. A sequence may seem perfectly acceptable to an untrained eye, when in fact it contains subtle artifacts wholly unacceptable in a training system.

Used properly, non-real-time emulations are a powerful tool in the development and testing of system concepts and algorithms. Despite the limitations cited, the emulations are extremely effective in providing a solid basis for hardware design. Thorough emulation will ensure that errors in system algorithms will be minor or at least confined to rare circumstances.

Circuit Simulation

As the system algorithms become defined, research workers interact with the systems engineering staff that prepares circuit card specifications. Questions of accuracy requirements emerge along with the design specifications, and additional analysis and emulation is required to resolve the issues. A typical example in image generator development is the data format and accuracy required for distance computations associated with the creation of texture and atmospheric fading effects in the imagery. At one point, nearly two dozen such issues were under active evaluation in a recent image generator design.

As the issues become resolved, circuit card specifications emerge and the implementation phase of development begins. Having used high level language emulations to build confidence in the system algorithms, the next emulation task is to ensure that the system and circuit designs correctly embody the algorithms.

This is done through gate-level logic and timing verification of the circuit designs. It is only within the past few years that design systems have emerged that feature gate-level circuit simulation as a part of a more general computer aided design process. There are now a number of vendors meeting the needs of an expanding community of users who must cope with systems of complexity that challenge traditional methodology [2].

Use of the design system begins with schematic capture, the entry of components and interconnections onto the system through an interactive graphics terminal. The logic and timing properties of each component must be described separately, and are kept in a library on the system. The design engineer must also define "test vectors" for the circuit; these are input combinations of ones and zeroes along with the expected outputs.

Test vectors are put into the design system, which uses the modeled components with the circuit interconnects to evaluate the logical operations. The engineer compares the computed outputs with those he has prepared separately. If the results do not agree, the most likely possibilities are that there was an error in inputing the schematic, an error in the circuit design, or an error in the construction of the test vectors. The error must be corrected, regardless of source, because the schematic will be used to drive the automated manufacture of the circuit card, and the test vectors will be used as part of the test of the fabricated card.

Timing verification is conducted along with logic verification to ensure that propagation delays do not result in failing or erratic performance. The design system checks timing over the component tolerances and the temperature extremes of operation. Signal delays due to interconnecting wires can be modelled with nominal values, and, if the situation is critical, rechecked with values derived from the manufacturing layouts.

To minimize fabrication errors, the list of components and interconnections used in the logic verification should be transferred without manual intervention to the manufacturing system. This must be done not only to avoid errors in manual transcription, but to ensure that all corrections are reflected in the logic simulation data base. The circuit card may be fabricated by wire-wrapping, printed circuitry, or other automated wiring techniques, but a computer interface will be required for each process.

Card Testing

Cards are tested individually to verify their functionality prior to incorporation in the new system. Conventional card testing equipment has the advantage of being adaptable to a wide variety of circuit card formats, but frequently lacks features useful in the debugging of new systems. For our image generation system, we wanted an approach inexpensive enough to permit construction of multiple test stands, capable of operating at the full clock rate (up to 40 MHz.) of the system, and compatible with the final implementation of diagnostics in the completed system. These requirements could be met only by working out the diagnosis and test philosophy well in advance of detailed design of the circuit cards.

The system under consideration here uses pipelined logic. On each clock signal, data is transfered out of a pipeline register, through combinational logic, and into the next pipeline register. By making each pipeline register a shift register, a technique sometimes called level-sensitive-scan design [3], it is possible to introduce data serially into any pipeline segment for test purposes. The results can be extracted from any later register by similar serial-operation of the registers in shift-register mode (Figure 3).

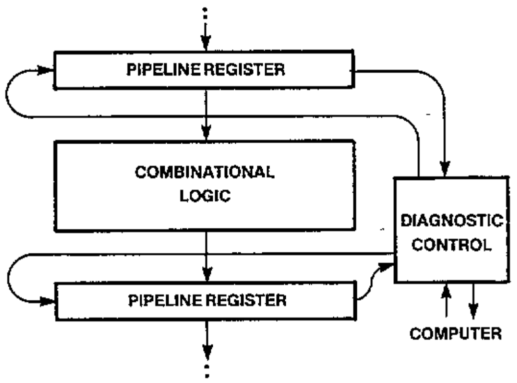

Figure 3. Diagnostics access every system register.

For uniformity of operation, a semi-custom logic chip was built to handle the diagnostic control. The designer incorporates one or more control chips on each board to interface the board with the system diagnostic card. The system diagnostic card, in turn, provides synchronization of the diagnostic functions, computations used for digital signatures, and interface to a computer that supplies test vectors.

For the test of individual boards, a microcomputer is used to provide the test vectors and to analyze the results. The board test station thus comprises the microcomputer, a system diagnostic card, and the card under test (Figure 4). Provision is made for the mounting of a second card under test, so that comparisons can be made between two cards of the same type. However, the initial source of test vector data will usually be from the circuit simulations used to validate the design prior to fabrication.

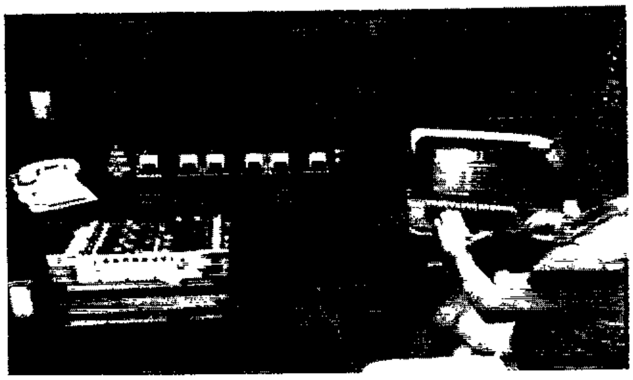

Figure 4. Microcomputer controlled test station.

The first experience with the test station was to check out the diagnostic card itself; this is possible because the card is designed for such self-diagnosis. Only five errors were found during the test of the four hundred chip card. Interestingly, three of the five errors were in the manufacturing software and its interface to the design system. One error was a mistake in the manual of one of the components that was incorporated in the design library. The remaining error was a genuine design error, in a part of the circuit "too simple to require verification"; so it had never been simulated.

The total check-out time for the board was under three weeks, including the debug of the test station, the microcomputer interface, and the diagnostic software. The diagnostic card in a previous system developed with traditional methodology required seven weeks to check out, despite its having only half the number of chips. Rapid test helps make up for extra time spent in design verification, but the real payoff is expected in system test.

System Test

Circuit cards emerge from test over a period of time, so that the final system must be built up gradually. Since the same diagnostic techniques are used in the final system as the test stand, it is possible to test cards in functionally related pairs or groups. Ideally, it is best to complete the output cards of a digital image generator first, because it will then be possible to observe the image as an additional diagnostic aid. The system may then be completed working back away from the display.

The test vector data will be available for each card, although rather than using the microcomputer of the test stand, testing will eventually transition to the system minicomputer. The diagnostic card interface is designed to be interfaced to either computer, and the diagnostic software, written in Fortran, is transportable.

Additional test vectors are prepared by the design engineers to test larger portions of the system, and experience indicates that relatively simple test patterns are usually sufficient for most testing. Nonetheless, it is possible to return to the system high-level-language emulations to extract test data of great variety and complexity. The most simple example is the display of an image from stored values of the color components of each pixel. An image may be downloaded pixel-by-pixel from the high-level-language emulations through the diagnostic channels to the output buffer of the image generator. The performance of the image generator's buffer, digital to analog conversion, and display interface can then be evaluated with a complex scene prior to completion of the whole image generator.

HARDWARE VS. SOFTWARE PROTOTYPING

Is all of this emulation and verification really neccessary? Do designers really make a lot of errors? In fact, designers make very few errors as a percentage of the design decisions they make. The problem is that as systems become more complex, that small percentage becomes more and more costly to fix. Moreover, there are many more possibilities for misconceptions rather than ordinary errors. As a system becomes larger, it is much easier to overlook all the implications of a design decision made in one small part of the system. The designer may well correctly implement the wrong function.

The potential for error rises at least geometrically with the number of elements in the system because of the multiple interactions involved. The situation seems worse in a special purpose device like an image generator than it does in a device like a general purpose computer, because all of the functions are coupled so closely. Moreover, the coupling makes it more difficult to isolate a particular error, so it tends to take longer to identify and fix each problem.

On top of the general complexity, the use of advanced packaging techniques makes traditional methods of hardware debugging more difficult. The extreme case is circuitry embedded in a custom or semi-custom integrated circuit, in which it is not only impossible to access all the signals, but any redesign of which, no matter how slight, requires an expensive and lengthy manufacturing process. Outside of custom chips, high speed logic requires controlled impedance circuit boards and attention to lead lengths, requirements that do not adapt well to easy redesign.

The net effect is that complexity increases do not dramatically affect the time it takes to build a traditional hardware prototype, assuming manpower is scaled according to the size of the project, but it very dramatically affects the time it takes to debug the prototype. At some point, the debug time becomes completely unthinkable and a software prototyping methodology is the only approach practical.

PROBLEM AREAS

The emulation methodology as described, with elements of high-level-language emulation, design verification, board test, and system build-up, has proved to be every bit as powerful as it sounds. But implementation of the methodology has not always been easy, and there are some special difficulties worth noting.

The high-level emulation system is totally new, and took several years to develop and perfect in house. The most serious limitation of the system is the long computation time (10 minutes per frame) required to make an image, which in turn limits the number of sequences that can be examined for anomalies. This leaves open the possibility that algorithms will later prove to have shortcomings in obscure circumstances.

Although many of the logic verification tools are now commercially available, there are still some of the aspects of a pioneering effort. For example, the available logic design systems do not provide complete component libraries or standard manufacturing interfaces, so these are left to the user to implement. In addition, the speed of execution of moderately priced systems limits their use, both in terms of circuit size and number of test cases. New products are continually being introduced, however, that promise to improve the price to performance ratio.

A significant drawback of the schematic capture entry of design data is that it potentially shifts work best performed by support groups to the design engineer. It seems best to share use of the system so that support personnel absorb the bulk of data entry work, while the design engineers use the system for design verification.

In addition to schematic capture, test vector generation and input in a tedious task. One avenue towards easing the test vector problem is to write higher level language programs that reflects the function of the circuitry on a must higher level than the gate level emulation. If this can be done efficiently, test vectors may be prepared in greater quantities and with much less tedium than manual methods.

To be successful, the methodology requires a lot of attention to interfaces and "tool-building" in general. The whole emulation activity tends to front-load the project by adding tasks prior to the actual construction of hardware. Because building hardware is such a tangible sign of progress, there is additional pressure to pass over the unaccustomed tool-building and emulation tasks to get things built more quickly. Consequently, it takes a lot of self-discipline to stick to the methodology in the early phases of the project.

Application of the self-discipline is not always clear-cut. It is possible to spend too long on emulation work, trying more and more cases in order to fully test the design. The tendency, of course, is quite the contrary. Nonetheless, it makes the question of how much emulation is enough a very real and debatable issue.

Most of these problems derive from the newness of the methodology, and are likely to be resolved as a backlog of experience builds up the required tools and a track record of advantages that outweigh problems.

Finally, it is important to point out that methodology is never a substitute for basic skills, experience, or creativity. There is nothing in the methodology that helps define what the function of a product should be, how to structure it, or how to make a practical design. The methodology only helps ensure that the system conforms to the concepts of it implementers. A poorly conceived product may be engineered with a thoroughgoing emulation methodology and meet its design objectives with great facility, but fail to meet the marketplaces needs because of function or cost. Nonetheless, the methodology can bring a well-conceived product to a timely fruition, and that is enough to justify the approach.

SUMMARY AND CONCLUSIONS

Increasing system complexity will force the adoption of rigorous, software-based design methodologies. The trend in that direction is evident, emerging in part from the needs of the semiconductor industry to design integrated circuits with very large numbers of components. The characteristics of extreme complexity, short schedules, and imprecise specification involving subjective judgements that are associated with digital image generators and other large training systems all combine to make an emulation methodology especially useful for system development.

The method presented in this paper includes four steps in the development of a system. First, system algorithms are conceived and emulated in a high-level-language to produce, in the case of an image generator, still pictures and animated sequences that demonstrate the output of the product as conceived. In the next step, algorithms are converted to circuit card specifications which are designed and verified with a logic and timing simulator. Then, cards are fabricated and tested independently using simulation data fed to the card through a microcomputer in a test stand. Finally, the system is built up incrementally with additional test data derived from the original emulation used to test major functions.

Our experience with the methodology is incomplete; it will take a few years of having a completed system in service to tell if the design is really as thoroughly proven as we intend it to be. However, I believe that designs would have had to have been substantially simplified, with accompanying compromises in performance, had a less rigorous development approach been used. It would have been virtually impossible with traditional methods of prototype refinement.

Despite increased complexity, a thoroughgoing methodology makes it practical to make increased use of semi-custom integrated circuitry, so that the size and cost of the system is reduced as well. My best estimate is that the system will use about half as many components as competitive systems having fewer features.

The continued use of emulation methodologies will require an increased understanding from all those involved. General management must understand the capital requirements and benefits of new emulation and circuit design equipment. Project management must set its sights on a longer term view of scheduling; early fabrication of hardware no longer equates to earlier completion of the project. Design engineers must learn and adapt to the new tools. And customers must learn new measures of progress in the completion of design work.

REFERENCES

1. Kidder, Tracy, The Soul of a New Machine, Little, Brown, and Co., 1981
2. Evanczuk, E., "Integrating the Engineer's Environment", Electronics, May 17, 1984, p. 121
3. Gutfreund, K., "Integrating the Approaches to Structured Design for Testability", VLSI Design, vol. iv, no. 6, October 1983, p. 34

ABOUT THE AUTHOR

Roy Latham is a member of the Scientific Staff at the Link Flight Simulation Division of the Singer Company, specializing in digital image generation. He received B.S.E.E. and B.S. Aeronautics and Astronautics degrees from M.I.T. in 1970, an M.S. in Applied Mathematics from Stony Brook in 1974, and an M.S. in Computer Science from the University of Santa Clara in 1983. Prior to joining Singer in 1978, he worked on the development and testing of aircraft navigation systems at Grumman Aerospace Corporation. At Singer, he has worked on image generation systems as a project engineer and in R&D. The author is a licensed professional engineer, is a U.S. Patent Agent, has authored six previous papers, and holds a patent in the field of navigation.