Technical Report CS76002-R

FORMAL DESCRIPTORS FOR HARDWARE SIMULATION

by

J.A.N. Lee and T. C. Brown

Language Research Laboratory
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

1 January 1976

The work reported herein was sponsored in part by the National Science Foundation Grant No. DCR74-18108.

ABSTRACT

This paper reviews the current status of an ongoing effort to develop a hardware description language which would be suitable for use as both a design tool and a documentor. Included in the requirements for this language would be the necessity for the language to function not only in many areas, such as automated design and verification, or testing and simulation, but also at many levels. That is, to range over such applications as circuit design at one end of a spectrum to the validation of systems configurations at the other end.

This paper views the language requirements from three points of view,

i) the subjective (human) elements usually associated with the syntactic features of the language,

ii) the minimal semantic elements to be provided and the structures (both program and data) which are necessary, and

iii) the features to be included in order to facilitate the formal verification of the conformance of the descriptor to preselected attributes.

The work described in this paper is based on continuing research regarding the nature of formal descriptor techniques, on their applicability to automated theorem proving and techniques for improving the teaching of computer related languages currently under way at Virginia Polytechnic Institute and State University.

INTRODUCTION

In the fall of 1973, a group of individuals were invited to participate in a workshop at Rutgers University, on the topic of computer description languages. This workshop was sponsored jointly by the IEEE Technical Committee on Computer Architecture (TCCA) and the Special Interest Group on Architecture (SIGARCH) of ACM. Whilst the workshop itself discussed many topics including languages for the description of Integrated Circuit chips and modules, experiences with using hardware description languages and a comparison of the "anatomy and physiology" of computer description languages, the major event for which the workshop is remembered, is an after hours meeting of the instigators of most of the well known computer description languages.

It was at this meeting that it was proposed that an effort be undertaken to develop a standard computer description language which could be used as a means of communication between designers and which would be susceptible to use as a design aid. Three distinct uses were identified:

(i) As an aid in describing hardware designs formally and hence as an aid to the more adequate designing of systems,

(ii) as a basis for continuing work on the development of a means for "automatically" verifying the correctness of a design, and

(iii) as a description for use in the simulation of the hardware so as to test its characteristics and performance.

In order to achieve this end, an informal organization was established named the Conference on Digital Hardware Languages (CDHL), an executive board was named and a convenor was appointed (J. Lipovski of the University of Florida, Gainesville, FL). In the years subsequent to that workshop, two other workshops have been held, one in Darmstadt, West Germany (1974) and the other in New York in 1975. Additionally, the "conference" has continued its developmental work under the leadership of Lipovski, by correspondence. Much of the work of the group has centered around establishing a base for the development of a new language. In the main this has taken the form of attempting to reach agreement on two issues: the set of primitives which should be included in the language (and by implication the level of the language) and the set of control structures to be provided.

At the New York workshop (held in conjunction with the First Annual International Symposium on Computer Hardware Description Languages and their Applications*) the progress of this work was reviewed in critical detail so that further work was entrusted to a group of only five contributors. It was revealed at this review session that many of the decisions made by the whole conference in the previous two years had been contradictory and further there seemed to be some resistance to viewing the design process in abstraction prior to the selection of any syntactic language forms.

This paper is partially the result of this attempted design

----

* Su, S.Y.H. and D.L. Dietmeyer, 1975 International Symp. on Computer Hardware Description Languages and Their Applications Proceedings, IEEE Computer Society, New York,

experience and partially the result of continuing studies at VPI&SU to improve the effectiveness of computer related languages for many different purposes including design aids and learning processes.

SUBJECTIVE CONSIDERATIONS

The initial considerations for any language invariably center on the syntactic forms which shall be used. That is, on the language forms which the user will utilize. Whilst we are not in agreement with this sequencing of language development, we present here some considerations for a hardware description language which are totally subjective. In time it is hoped that some techniques for evaluating these measures can be produced.

Already some work has been initiated to measure the effects of certain language forms* and opinions have been expressed as to what is suitable for inclusion in languages. From the point of view of a Descriptor Language, it is necessary that a close examination be made of the potential clientele. It has already been established that Hardware Description Languages have a number of uses; as a design aid ,as a design verification input and as a means of simulating the characteristics of the design. Within each of these uses there may exist a wide band of user potentials from the design specialist who requires a language which is close enough to his logical thinking processes so as to not be a distraction, to the occasional user of the description as a means of verifying procurement specifications.

It is important, at this stage, to distinguish between the criteria for judging the purely syntax related elements of

----

* see Weinberg, G., The Psychology of Programming, Van Nostrand Reinhold Co., New York, 1971. and Shneiderman, B., and Richard Mayer, Towards a Cognitive Model of Programmer

the language from its supplied operational features. Sterling* reports on a workshop held at Stanley House** which developed the criteria for humanizing information systems. This workshop specified five elements as part of their criteria:

a) Procedures for dealing with users

b) Procedures for dealing with exceptions

c) Action of the system with respet to information

d) The problem of privacy

e) Guidelines for system design having a bearing on ethics.

Whilst the last three of these criteria are peculiar to information systems, the other two are generally applicable to languages for man-machine communication. However, of the thirteen requirements within these two categories only one refers to the language forms:

"The language of the system should be easy to understand."

Each of the other criteria refer to the operational characteristics of the system. These criteria might best be considered in relation with our section of semantics; however, since they are not related directly to the features provided by the system but simply to man-machine features* we shall omit further consideration of the Stanley House

----

* Sterling, T.D., Humanizing Computerized Information Systems,Science, Vol. 190, No. 4220, 19 Dec 1975.
** Sterling, T.D., Guidelines for Humanizing Computerized Information Systems, Comm.ACM, Vol. 17, No. 11, pp609-613,1974
* Transactions with the system should be courteous,
  A system should be quick to react,
  A system shoud respond quickly to users if it is unable to resolve its intended procedure,
  A system should relieve the users of unnecessary chores,
  A system should include provisions for corrections,
  [illegible] should be held responsible for mismanagement,

criteria.

Lipovski (in cooperation with Barbacci, Piloty and Lee) tabulated the ranges over which subjective language measures might vary; in many respects his measures are merely guidelines to language development and may by themselves be too leading to be acceptable to the whole community.

These criteria are based on requirements initially established by Barbacci* which he classified differently than we choose to do. That is, Barbacci classified his subjective criteria with respect to "the requirements for a scientific notation used in a design process (general properties); and ...(the properties of) the objects which we are devising (specific properties)." However, this classification schema cuts across the classifications used here, covering both syntactic and semantic qualities.

There are two views of a language which must be considered; the language from the point of view of the originator of a description and that from the viewpoint of the recipient or user of the the description. That is, the user of the language and the user of the message. Let us not presume to map these into a single criterion; rather we should be aware that what may be "simple" (a term which is yet to be

----

A system should recognize as much as possible that it deals with different classes of individuals,
A system should recognize that special considerations might occur that require special actions by it,
A system must allow for alternatives in input and processing,
A system should give individuals choices on how to deal with it,
A procedure must exist to override the system.

* Barbacci,M.R., A Comparison of Register Transfer Languages ..., IEEE Trans. on Computers, Vol. C-24, No. 2, February

defined) for a language user may not be as clear to the recipient. Hence the language should be both (relatively) easy to use and understand. Unfortunately these criteria refer to different users and possibly users with differing backgrounds. Obviously the originator of a descritpion has a vastly different concept of the device he is explaining than the person for whom the description is intended. Take for example, the understanding of a machine language from two similar points of view. That is, the understanding of a machine language by a programmer who was initially exposed to the architecture of the machine and the understanding possessed by a programmer who views machine language merely as the underpinnings of a high level language, are widely different. Experience shows that even their styles of programming is different, the machine oriented programmer making more use of the subtleties of the machine organization and being far more aware of the advantages that may be taken of the timing of certain actions.

It is difficult to describe a language in exactly the terms that Barbacci used and not recognize that the criteria overlap. Obviously the criteria of writeability and readability are interrelated. However, that feature which may be suited to writing is not necessarily the best criterion for reading. Take, for example, the problem of conciseness. At one extreme, conciseness assists the programmer in that he can express with the minimum of effort complex operations; that same conciseness can equally well obscure the purpose of the program or description from the reader. In the past, there has been much attention paid to the idea of writing programs in "English" or some subset thereof. COBOL attempted this in the name of "self documenting programs" which became a nightmare for the programmers. The keypunching of such verbose* programs alone accounts for the majority of errors; consider that the average programmer is not an expert typist and can rarely type more than 25 characters without an error. On the other hand, the reception of a program written in a verbose form makes the understanding of that program more easily ascertained by the reader. Hence there have been developed short cuts to COBOL style programming by which the programmer can be as concise or cryptic as he desires and subsequently a preprocessor expands this code into the more verbose form. Lipovski has suggested that the acceptable criterion for this quality lies somewhere in the range:

cryptic - concise - prosaic - verbose

where the natural languages would be classified as verbose and a language such as APL would be cryptic.

Where a new language is being proposed (as in the case of the consensus language of CDHL) it is essential that the language be acceptable to the domain of potential users. This requires the language to have three essential qualities; it should be readily learnt, it should be in a style which is familiar and it should be patently applicable to the problem in hand.

Language styles are important to the potential user; if the style of language permits him to recognize familiar

----

* Lipovski decribes COBOL as being prosaic (that is, is like

constructs and those constructs have meanings which match those of the prior use of a similar language, then the transition to the new language is considerably enhanced. The programming language literature abounds with references to "Algol-like" or "Fortran-like" languages without a clear definition of what these terms indicate. Similarly, in natural languages it is possible to recognize a language not from a knowledge of either its vocabulary or grammar, but simply from its "style". However, a syntactic style can mask semantics and the use of familiar syntactic (or even lexical) forms for new and unrelated operational purposes will lead only to confusion and hence to a resistance to learning.

On the positive side, consider the number of students now being introduced to programming through the use of BASIC. Whilst there is a (unfortunate, in our opinion) trend towards extending BASIC to encompass the features of other languages, such as file management, string manipulation and graphics, it has been shown that these programmers can easily assimilate the concepts of FORTRAN and PL/I. However, the programmer who has been weaned on APL finds the transition to be overwhelming even though the power of APL outstrips that of BASIC by several orders of magnitude. The applicability of a language (which is really the applicability of its operational facilities rather than its syntactic forms) is a measure of the ease with which commonly used solution elements can be expressed in the language. There exist two extremes of this quality. If the language is primitive (tending toward the abstract notion of a machine) then it is possible to show that anything is computable; however, the amount of effort that must be expended to express a solution or to describe a situation may be unacceptable or even impossible. At the other extreme, where a language has been "humanized" in the manner of Sterling, too often the applicability of the language has been reduced to the point where only one class of applications is possible. Take, for example, a language such as LISP which obviously has all features necessary to support the claim that it can compute all functions ; that is, the language contains the minimal set of operators and data structures necessary to simulate a Turing machine. But LISP is much more powerful than simply a Turing machine; it contains features which have permitted programmers to encode many esoteric artificial intelligence projects which are landmarks in programming history. Yet LISP would not be best suited to many data processing applications and by no means would it be suited to the description of hardware. The special purpose languages, developed to be of the greatest use to certain specialized segments of industry, such as COGO (for surveyors) and STRESS (for structural engineers) or the various statistical application packages such as BIOMED, would not be applicable in our sense of the quality. Equally, the generality of a language can be a burden when a restricted range of problems is to be solved. Of equal concern here should be the failing that a language which is intended for use as either a descriptor or a simulation may dwell too much on the niceties of description or simulation to the detriment of the purported goal, in the same manner as a machine language can be overly close to the machine being used. That is, a language may be developed which is convenient to express the concepts of the process of description or of simulation and thus be one too many steps away from the desired level at which the description or the simulation is to be written.

As was expressed at the Stanley House workshop, a system should relieve the user of any unnecessary chores. Such chores should include at least such features as not worrying about the operating system within which the simulation is to be run or the description is to be verified. Most annoying of many systems today is the claim of "system transparency" which apparently means that the user must see through the system to the underlying support features. We would prefer to use the term "transparent" to indicate that such machine or operating system features are not seen by the user. Thus in contrast to the Stanley House criterion it would be more appropriate here to suggest that the language (and its host system) not require the user to be aware of features which are not essential to the description or simulation.

Finally there are two interrelated properties which are necessary. Whilst the syntax of a language must be powerful enough to express complex concepts in a concise manner, the syntax must be well enough organized so as to develop a logical relationship between the elements of a description. That is, language elements which have similar purposes should have similar constructs. For example, if a language is to contain several means of (say) output there ought to be some commonality between the various forms so as to ease the understanding of the language elements and better to eliminate confusion between the forms which possess operational commonality. In the extreme, the commonality between features may result in a single syntactic form such as has been developed for APL. In such a system the need for conformance to a single syntax restricts the use of meaningful mnenonics or (as in APL) to the introduction of special character instances to represent certain operations. A language such as PL/I on the other hand is a conglomerate of forms, any attempt at conformance being tenuous.

Thus whilst syntactic simplicity may be subject to some quantification, such as by measuring the maximum depth of a derivation tree necessary to produce common forms of sentences, semantic simplicity may have a converse effect. The simplicity of semantic features, as expressed earlier, may result in the need for complex implementation of common feaures. However, the presence of semantic simplicity combined with the ability to introduce named functions or procedures, especially when such commonly used procedures or functions are available in a readily available library, provides a compromise which is acceptable.

SEMANTIC REQUIREMENTS

The subjective qualities described in the previous section are applicable to computer languages in general; however it is the operational characteristics of a language which set it apart from the total set of languages. Provided that a language has met the subjective requirements then its potential user is ready to investigate its semantic characteristics further.

In the particular case of hardware description languages, there has been a need for specialized languages recognized for some considerable time. In a survey of the characteristics of existing languages* it was observed that the early languages were described first in 1964 and that there has been a steady proliferation since that point in time. Of these languages only a few have actually been implemented either as a language by which a system can be directly modelled or as a descriptor which can be used as the basis for a simulation. That is, some languages have been restricted in their usage to simply human-to-human communication, such as the manner that ALGOL was initially proposed (and used for several years in Comm. ACM.)

Following considerable experience in describing machine architectures the Language Research group at VPI&SU, reviewed the needs which had not been met in several (nine)

----

* Lee, J.A.N., et al, ibid

different, but well estblished languages. At this point it is important to note that this survey was conducted in the light of efforts to describe not complete systems, but instead simple architectures such as that of the DEC PDP-8, the PDP-11, Foster's "Edinburg Machine" and his stack oriented computer "SOCRATES", and the microprocesor described by Weisbecker# amongst others. In each of the attempted description generations certain shortcomings of the languages were observed with respect to two elements; either it was impossible to model a certain feature of the prototype, or no facility existed to verify the correctness of the description of a design or more importantly, the design itself.

Four distinct semantic requirements were established as being necessary for inclusion in the minimally acceptable language;

facilities for the description of

concurrency of actions

the timing of events

and the hierarchy of both functions and components

and the provision of facilities to verify the integrity of the components of the prototype.

Surprisingly enough, of the nine language surveyed (DDL, CDL, LOTIS, ISP, CASSANDRE, CASD, RTL, LOGAL and AHPL) not

----

Weisbecker,J., A Practical Low-Cost Home/School [illegible] System, IEEE Computer, August 1974, pp. 20-31

one satisfied all the established criteria. That is, whilst all but one (CASD) had at least some elementary facility to provide the execution of concurrent processes, only one (LOTIS) provided for the lock out of processes from the gating of an already operational elements such as the memory or a register being used by some subprocess. However, it is necessary that we point out that LOTIS has not yet (since its original publication in 1964) been implemented. A further requirement is the inclusion of a facility to describe the actual data paths between functional elements. That is, to provide a means by which data path utilization can be checked. In at least one effort to describe a processor, the absence of a data path between two registers was identified only after considerable effort had been expended (manually) to determine that two concurrent operations did not use the same bus. In the majority of existing hardware description languages, the movement of data between elements was assumed to be unimpeded. Only when a descriptor is given the facility to know which paths (which may contain several segments) are utilized in register to register data transfers, can the integrity of data paths be verified.

The timing of events within the description of a processor would seem to be of such fundamental importance, that every description language would provide the barest of facilities. Obviously, where nothing more than sequential (non-concurrent) events are permitted, then there is no need for any special timing facilities. Whilst every description system surveyed provided asynchronous timing facilities, synchronous operations were not as well supported. In the main, where synchronous actions were provided, this was accomplished by means of a pulse identification technique. That is, actions were labelled by identifiers, the commonality of identifiers establishing the correspondence of timed events. Conversely, the actual timing (with respect to some clock) of the duration of events is provided in only two systems; LOTIS and CASD. In these systems the number of cycles necessary to complete an operation may be specified and other actions may be predicated on the amount of time elapsed since a specified event occurred.

In summary then, the minimum semantic features to be provided by a hardware description language should include:

concurrency

synchronization by pulse identification and by cycle counting

asynchronous operations

integrity of operations by the lock out of secondary activities and the preservation of the sole use of data paths,

and the ability to describe a hierarchy of components and functions so as to provide the means for developing descriptions in a top-down manner and to provide the capacity for describing intrinsic elements which are to be used commonly.

The following table (taken from Lee et al) indicates the lack of facilities in existing hardware description languages:

			DDL	CDL	LOTIS	ISP	CASSANDRE	CASD	RTL	LOGAL	AHPL
CONCURRENCY			Y	Y	Y	Y	Y	Y	Y	N	Y
TIMING	Synch by	pulse id	Y	Y	Y	Y	Y	N	Y	Y	Y
		counting	Y	N	Y	N	N	N	N	N	Y
	Asynchronous		Y	Y	Y	Y	Y	Y	Y	Y	Y
INTEGRITY BY	Locking out		N	N	Y	N	N	Y	N	N	N
	Data Path Def.		Y	I	I	I	C	I	I	I	C
HIERARCHY OF	Functions		Y	Y	Y	Y	Y	Y	N	Y	Y
	Components		Y	Y	Y	Y	Y	Y	N	N	Y

Legend:

Y - yes

N - no

I - implied

C - claimed but no examples given

VERIFICATION ELEMENTS

This section describes the kinds of verification which a hardware description requires and relates these to the state of the art in program verification research.

From the subjective requirement that a hardware description be easy to understand it follows that the description should be hierarchical, with a natural distinction between functional units and their realization at the next lower level of description. This requirement suggests two distinct stages of verification:

I. Verification that a given level of description satisfies specified behavioral properties (input-output relations, preservation of invariant relations, deadlock-free, etc.).

II. Verification that the functional requirements of one level of description are satisfied by the next lower level of description.

Stage I is directed toward proving that the system under consideration conforms to the designer's behavioral intentions. Stage II is directed toward proving that the functional r quirements of a description element are realized by a corresponding system at the next lower level. These two stages are consdidered separately below in order to more clearly assess their impact on hardware description language design.

Stage I Verification

Input to the Stage I verifier will consist of a system description and a specification. The description will consist in essence of a system of typed variables and concurrent processes operating on these variables subject to synchronization constraints expressed or implied. The specification will be a system of input-output relations defined on certain system variables--one for each operator or process of the description. Output of the Stage I Verifier will consist of some or all of the following:

(i) a proof that the system description satisfies its specification, or a counterexample;

(ii) a request for more detailed specifications for designated system components;

(iii) a proof that the system is deadlock free; and

(iv) an analysis of which variables are local to a strictly sequential process and which are potentially shared by two or more concurrent processes.

Hardware semantics was found to entail several forms of concurrent process synchronization which the description and specification language must be able to express and the Stage I verifier must be able to analyze:

(i) Synchronization by event (e.g., clock pulse identification) and by time interval (pulse counting).

(ii) A shared variable may be concurrently accessed by processes which do not reset it.

(iii) A shared variable must be sequentially (exclusively) accessed by a process or operator which may reset it.

(iv) Certain concurrent processes may have global priority over others (e.g.,interrupts or hardware ON-conditions).

Hardware semantics does not entail a description-language control structure which mimicks the object-machine control structure: the existence of branch instructions at the micro-program or machine instruction level does not necessitate the existence of go-to statements in the description language, whose procedural component exists simply to define processes over the system variables.

The following paragraphs illustrate how the above requirements can be fulfilled by a description and specification language which is consistent with the state of the art in formal program verification.

Abstract data types. A given level of description contains a set of primitive types (either atomic or defined at a lower level) and a set of abstract data types defined on the primitive basis. An abstract data type, or data abstraction * names a system of selectors and operators, including a designated initialization operator. The abstract type includes a specification for each of these, but not their definitions as functions or procedures. Assignment may or

----

* Liskov, B.H., and S.N. Zilles, Specification Techniques for Data Abstractions, IEEE Trans. on Software Engineering 1 (March 1975),7-19.

may not be an allowed operator for a given type.

Thus, the basic data transfer statement is the action of an operator on a variable of its type.

Verification of a description containing abstract data types is now a well understood process: one assigns a conjectured invariant relation I(x) to an object x of type T and then verifies that each operation either establishes or preserves I(x) . A correct and sufficient choice of I(x) for a given Stage I verification task remains a difficult search problem, and may require user guidance (as in ojtput (ii) of the Verifier).

Sequential control. The Bohm-Jacopini system consisting of sequential composition (S1;S2), conditional (If B then S1 else S2), and repitition (While B do S) has an adequate system of verification rules based on Hoare's axiomatic approach. However, finding an adequate loop invariant for (While B do S) can be a difficult search problem in practice, requiring a combination of mechanical analysis and human guidance. *

Concurrency and synchronization. Hardware systems have a fixed number of concurrently operating elements. The Cobegin S1,...,Sn Coend construction is therefore appropriate for describing the concurrent processes realized by such elements. Several solutions to the mutual exclusion and synchronization problems have been proposed . A recent milestone in the analysis of concurrent processes has been the demonstration by Owicki * that a program verification

----

* Wegbreit, B.,The Synthesis of Loop Predicates, Comm. ACM (February 1974), 102-112.

calculus for RPL, a language with the above sequential control statements augmented by Cobegin...Coend and conditional critical regions, is consistent with a VDL-styled operational semantics and complete relative to any axiomatization of the language's data structures.

It is apparent that the results for RPL could be extended to a hardware description language with conditional critical regions for shared variables. Concurrent and sequential access to a variable could be modelled by a corresponding built-in semaphore (for concurrent access) which would be blocked from further incrementation by a nonempty request queue for sequential access to the same variable. A (Whenever B do S) statement could model an interrupt triggered by event B throughout its containing block; this is equivalent to inserting (While B do S) between each pair of statements in the block.

Recursion. For sequential programs, verification of recursively defined procedures can be accomplished by complete computational induction and similar methods *. Verification of recursively defined data types can be accomplished similarly by structural induction. However, adequate verification methods for recursive procedures containing Cobegin...Coend statements have not yet been developed. Moreover, it is doubtful that either recursive concurrent procedures or recursive data types are essential

---

* Owicki, S. S., Axiomatic Proof Techniques for Parallel Programs., Doctoral Thesis,(Department of Computer Science, Cornell University, July 1975).

* Manna, Z., Mathematical Theory of Computation, (McGraw-Hill, 1974).

for hardware description.

Stage II Verification

Verification that a set of selector and operator definitions satisfy the specifications of an abstract data type is a straightforward application of Stage I verification methods. However, the synchronization operations of the higher level must be enforced by the next lower level. This could be most easily done by directly modelling the semaphores and finite queues built into the description language semantics. It is in the synchronization area that the description language can be expected to exert the most profound influence on hardware design.

CONCLUSIONS

Our principle conclusion is that syntax, semantics, and verification elements must all be considered in the design of a hardware description and simulation language. Recent research * has made it apparent that an adequate hardware design language must encompass several levels of description and must have powerful facilities for verifying that one level correctly simulates the next higher level. These facilities will probablly be interactive, combining both program verification methods and algebraic simulation theory.

----

* Leeman, Jr.,G.B., W.C. Carter, and A. Birman, "Some Techniques for Microprogram Validation," Information Processing 74 (North Holland, 1974), 76-80

REFERENCES

Barbacci,M.R., A Comparison of Register Transfer Languages ..., IEEE Trans. on Computers, Vol. C-24, No. 2, February 1975.

Lee, J.A.N., D. Macock, P. Marks, T.C.Wesselkamper, The Requirements for Effective Hardware Description Languages, Language Research Lab., Dept. of Computer Science Report CS 75011-R, VPI&SU, Blacksburg, VA, December 1975.

Leeman,Jr.,G.B., W.C.Carter and A. Birman, Some Techniques for Microprogram Validation, Information Processing 74, North Holland Pub. Co., 1974, pp 76-80.

Lipovski, J., Memorandum 3.1, Conference on Digital Hardware Languages, (Internal Memorandum), 1975 October 10, University of Florida, Gainesville, FL. (to be published).

Liskov, B.H., and S.N. Zilles, Specification Techniques for Data Abstractions, IEEE Trans. on Software Engineering SE-1

Manna,Z., Mathematical Theory of Computation, McGraw-Hill Co., New York, 1974.

Martin, J.J., Some Theoretical Foundations of Program Testing, Dept. of Computer Science Report CS75029-R, VPI&SU, Blacksburg, VA, December 1975.

Owicki, Susan Speer, Axiomatic Proof Techniques for Parallel Programs, Doctoral Thesis (Dept. of Computer Science, Cornell University, July 1975)

Shneiderman,B., and R. Mayer, Towards a Cognitive Model of Programmer Behavior, Report TR37, Dept. of Computer Science, Indiana Univ., August 1975.

Sterling, T.D., Humanizing Computerized Information

Systems, SCIENCE, Vol. 190, No. 4220, 19 December 1975.

-------------, Guidelines for Humanizing Computerized Information Systems, Comm. ACM, Vol. 17, No. 11, pp 609-613, November, 1974.

Su, S.Y.U. and D.L.Dietmeyer, 1975 International Symposium on Computer Hardware Description Languages and their Applications, Proceedings, IEEE Computer Society, New York, 1975.

Wegbreit, B., The Synthesis of Loop Predicates, Comm. ACM 17 (February 1974),pp. 102-112.

Weinberg,G., The Psychology of Programming, Van Nostrand Reinhold Pub. Co., New York, 1971.

Weisbecker,J., A Practical Low-Cost Home/School Microprocessor System, IEEE Computer, August 1974, pp 20-31.