A methodology for design of digital systems—Supported by SARA at the age of one*

by GERALD ESTRIN

University of California at Los Angeles

Los Angeles, California

INTRODUCTION

There are many practical forces at work which encourage improvement in design methods. "Structured programming," "hierarchical design" and "modularity" are terms which have become commonplace. However the frequency of their use does not mean that design methods now exist which can guarantee that a particular complex system, whose programs are shared by many users, will have no design errors. Nor do design methods yet exist which can guarantee that a digital system, "simple" enough to be integrated into one device and reproduced in the millions, will have no design errors. It is well-known that almost all digital systems have so large a set of possible sequences of states that great ingenuity must be applied to achieve any meaningful degree of partial verification of their behavior. Theoretical advances continue to be made in verification. Those deriving from concepts of abstract data types\(^1\) are incorporated in new languages\(^2-4\) and have a lasting influence on the way we think about systems.

Despite those advances it is a source of great embarrassment to computer architects that semiconductor technology has created a capability which we cannot now exploit with proven design methods. A number of interesting approaches to design of concurrent systems are being actively pursued. However we do not currently have the languages, operating systems and design methods needed to effectively employ the new LSI devices which can now be produced. We would like to be limited only by economic factors, not technical or theoretical factors.

One hope is that there will evolve effective methods for composing such systems from defined, well behaved building blocks whose composite behavior can be shown to meet prestated requirements. A second hope is that starting from well formulated requirements and an initial abstract solution system, there will evolve helpful guidelines for structural partition and effective algorithms for behavioral refinement. The refinement procedure should conserve desirable properties through as many levels of abstraction as the design needs. A third hope is that a new dimension for architecture of computer systems will emerge from these design methods and permit us to effectively use the outpouring of large scale integrated devices.

Realization of these hopes and their joint use are the goal of the methodology discussed in the first part of this paper. The second part of the paper describes the state of the system implemented to support a designer attempting to use the methodology. A companion paper\(^5\) illustrates application of our methods to design of software.

The System ARchitects Apprentice, SARA, is only one year old. The roots of this work were reviewed in short by the author in one of a set of presentations\(^6\) which announced SARA early in 1977.

We are not so simplistic as to think that we have THE only right way to design computer systems. We do feel that our methodology makes some significant advance in approach to design of concurrent software and hardware systems and has great promise for incorporation of previously verified models of software or hardware systems. In fact it has long been recognized that methods for design of concurrent systems are also important to good sequential system design. The models used to represent desired concurrency can also be used to model a set of processes which are mutually independent. Sequence can then be imposed on them after appropriate analysis rather than by chance.

As verification methods become more effective and succeed in avoiding combinational explosion\(^7\) we hope to combine them with the UCLA design methodology and move closer to effective use of semiconductor device capability.

The UCLA design methodology

We tersely characterize the UCLA design methodology as being requirement-driven and supportive of self-documenting design of modular, concurrent, hardware and software systems. Figure 1 displays a high level flow chart of the design procedure. We will proceed through each step discussing what is done with particular attention to implied analysis or value judgments made outside of the stated procedure. In this first part of the paper the reader need make no assumption about the nature of automation aids but rather should focus on the systematics.

* This research was supported by the U.S. Department of Energy, Contract No. EY-76-S-03-0034, PA 214.
Figure 1—UCLA design methodology
**Initialization of a design**

This design methodology is characterized as requirement driven. Given a set of requirements and assumptions about the environment, the designer conceives of a system to satisfy those requirements. The bulk of our discussion is about a procedure which develops sufficient detail about the system behavior to determine which of the requirements are not satisfied.

Requirements are not God-given and there continues to be extensive study of disciplined analysis techniques leading to well formed requirements. When design costs get too high it is often necessary to reconsider the requirements themselves. For purposes of this paper we will assume that requirements have been carefully generated and that we are to make our best effort to design a system which satisfies them. It is generally recommended that requirements be made at a sufficiently abstract level that they do not prematurely commit to particular implementation methods. However, many designers make certain implementation decisions early for practical reasons. For example, a manufacturer may restrict implementation to a small prescribed set of well tested, well understood building blocks, in order to minimize inventory and training costs. These must then be carried through the rest of the design process and not be compromised.

Requirements fall into different classes with respect to their use in this design procedure.

Some requirements carry quantitative design constraints, such as real time response to a stimulus, which lead directly to use of analytical methods or to generation of experimental tests. These are the most straightforward requirements to deal with, with the exception of two problems:

- A numerically stated requirement often implies more precision than was intended and it is sometimes tedious and difficult to introduce tolerances or to manage tolerances during evaluation.
- When a high level system design is partitioned it is necessary to determine the set of subsystems which are involved in satisfying a quantitative requirement and, where possible, to estimate a new quantitative subrequirement for each subsystem. Such estimates can turn out to be quite arbitrary and to produce undesirable distortions of the design. This is a poorly understood art.

Many requirements are qualitative. In some cases their satisfaction is determined by delegating the responsibility for judgment to individuals who can then record approval or disapproval. In other cases qualitative requirements are simply passed on to the fabrication phase because they cannot be evaluated until manufacture. Qualitative requirements deserve careful attention because they are often the focus of user dissatisfaction. When tradeoff analysis can be made to show optimality of a design response to a qualitative requirement it can save a great deal of misunderstanding and expense.

An integral part of our design methodology requires initial behavioral modeling of the system under design and its assumed environment. The environment model displays only those properties necessary to constrain the system design and is constructed using the same primitives as those used for models of systems being designed. Therefore when two independent designs are composed, a designer must test the consistency between each subsystem and the others' environment, as informally shown in Figure 2.

**Partition or composition decision**

The decision to compose a model of the system directly out of predefined models or to go through another stage of top down partition depends upon the availability of predefined building blocks and the designers understanding of their behavior. A designer may have a number of building block models, which seem to fit a particular design jigsaw, but be missing one type of building block. Rather than pass through a uniform top-down partition, which always entails some overhead, the designer may choose to move aside into
an independent design phase. There the designer would establish requirements for the new building block, successfully complete its design and validation using existing building blocks, and then return to complete composition along the original design path.

It is of value to add to our intuitive acceptance of what building blocks are by the following definitions:

By building blocks we mean physically realized systems with:

- constrained environments, and
- predictable input-output behavior,

such that, given:

- two or more such compatible systems, and
- connectivity constraints between them,

then the combined input-output behavior is predictable. Whenever a combined system is also a building block we call it well behaved.

Building block models are representations of building blocks if:

- each model has one or more defined physical realizations satisfying the definition of a building block, and
- when building block models are interconnected consistent with interconnection constraints, then combined corresponding physical realizations form a physical realization of the combined model.

Composition

In the composition step the full set of selected building block models in interconnected—to each other and to the assumed environment model. Attributes, associated with the inputs and outputs of each building block model, provide the basis for consistency and completeness checks. These checks must include an environment check as noted above for Figure 2. The complete model is then analyzed or experiments are conducted in a simulation environment.

If implementation decisions have not been made earlier they are introduced in the composition step building block models.

Composition evaluation

The evaluation step consists of a combination of analysis and experiment. If the behavior of each building block has been verified, it may be possible to verify a system requirement which has been formally stated. For example, if one of the requirements states that a concurrent system must satisfy “proper termination” then it is possible that analytical tools\textsuperscript{16,17} can be used. More generally the designer will generate experiments in a simulation environment and choose the most stressful conditions for test.

Partition

A partition procedure is used when a set of requirements cannot be met directly by composing known building blocks. There are different strategies for guiding partition of a system into subsystem modules.

- Parnas\textsuperscript{18} has noted that modular decomposition may be done according to independent user functions which need to be implemented or according to system functions which are needed in carrying out every user function. The former approach may make mapping of requirements more direct. The latter approach generally leads to better utilization of shared resources.
- One module may be separated from another because both design and observation of the included operations are considered vital to performance, reliability or cost. The module is a system element which is most easily observed because its interface with the outside is made explicit.
- A set of modules may be selected because it is estimated that they all have about the same complexity, thereby providing a design balance.
- A set of modules may be selected because it is estimated that they all have approximately equal influence on performance and therefore represent an educated guess of a balanced system.
- Modules may be selected because they are readily composed from known building blocks and therefore will avoid another partition step.
- Modules may be selected because they simplify refinement of higher level behavioral models in a manner which guarantees preservation of desired properties. Thus, consistency with invariants established in higher level proofs of data flow or control flow properties would be example criteria.

Whichever combination of guidelines are practiced leads to the creation of two or more structures and their interconnection. For each named subsystem there is then an initialization step identical to that carried out at the previous level with the exception that the higher level requirements must be mapped onto the new structures. For each named subsystem a behavioral model is developed which seeks to meet the subsystem requirements and hopefully to join in meeting full system requirements.

Partition evaluation

The set of subsystem models are evaluated by inspection, by analysis or by simulation. If any evaluation fails, the design is returned to the partition phase for modification or re-creation by the designer. If all evaluation steps pass then the design is returned to the part of the process where, for each subsystem, it is decided that composition can take place directly or else that another partition is needed.
Implementation

When the compositions succeed in passing all tests for all subsystems, the design moves into an implementation phase to prepare the design for fabrication. The following operations are generally needed:

- Documentation is prepared. If there have been careful records kept during the design process, they provide the best description of intended functions, predicted behavior, and the usually critical interfaces between subsystems. This documentation is important for those who will evaluate the fabricated system. However other kinds of documentation are generally needed to facilitate fabrication, maintenance, marketing, training and distribution. Wiring lists, microprograms, bit maps for read-only memories, higher level language programs and test programs are examples.
- Changes may have to be made to remove artifact introduced by the modeling and design process. For example, instrumentation may have been integrated with a model to determine whether performance requirements were met but the measurements may themselves deteriorate performance and not be deemed necessary. This excision must be done very carefully in order not to alter desired behavior and sometimes it will force another pass through evaluation of the altered model.
- The simulation environment may have mechanisms which cannot be directly fabricated and are replaced by “equivalent” mechanisms. This too must be done very carefully lest the modeling and evaluation process be invalidated.

User interface

All communication between a designer and the SARA DESIGN SYSTEM as well as output to FABRICATION are done through IO. The IO system creates a flexible and powerful user-interface and achieves uniformity of interface with the various tools in SARA. Moreover it serves as a place to contain all machine dependent routines, easing any move from one PL/I environment to another. SELECTOR is then used to move between language processors at different levels as illustrated in Figure 5.

Structural and behavioral modeling tools

The most fully tested tools are those contained in the modules named STRUCTURES and BEHAVIORS.

- STRUCTURES holds a language processor called SL.I which lets a designer interactively describe the structure of a system using three primitives (see Table I): modules, sockets and interconnections. Multilevel structural models can be constructed using fully nested modules as in Figure 6. Multilevel interconnections, can be expressed simply, as displayed in Figure 7. In general interconnections may be refined as complete systems.
- BEHAVIORS holds several language processors used to create different kinds of models of named structures. The principal behavioral modeling tool is the Graph Model of Behavior (GMB). The GMB provides a small number of primitives (see Table II). The control graph primitives allow a designer to interactively create a graph explicitly modeling flow-of-control among processes including synchronization of concurrent processes. The data graph primitives allow a designer to display flow-of-data, interpreting the processes in the control graph at an abstract level. Furthermore a preprocessor, called PLII allows a designer to concretely interpret each data processor and dataset primitive using the full power of PLI. The topology of both the control and data graphs is fixed and they are used to model pseudo static aspects of system behavior. The PLI interpretation (PLIP) models dynamic behavior within each processor and causes changes in control and data flow along the prescribed data and control arcs. Two other tools have been built for analysis of the GMB model. An interactive simulation environment permits experiments to be executed, observed and analyzed. The simulation environment is used to check predictions of results and estimate timing. Moreover, the GMB simulator is effective at detecting deadlocks, detecting contention for datasets and detecting contention for processors. One reduction program is used to analyze the control flow graph and this analysis can determine that a system is "properly terminating", i.e., that it is deadlock free and reentrant.

THE SARA (SYSTEM ARCHITECTS APPRENTICE) SYSTEM

The groundwork for SARA was set by Gardner* at the beginning of 1975 following several years of collaboration between Gardner, Potash and Estrin. The design of SARA was begun during the summer of 1975. Implementation of structural and behavioral modeling tools continues through the present. SARA's birth was announced at the beginning of 1977 and a one year old version now marks SARA-reborn at MIT-Multics, a more permissive environment than UCLA’s IBM/TSO. The infant version of SARA provides a collection of modeling tools, at both UCLA and MIT, for which some care has been applied to the design of the user interface. The sibling SARA's are reachable by authorized users through the ARPANET.

The current view of the structure of the SARA system is shown in Figures 3 and 4.

* In addition to R. L. Gardner, major contributions to the design and implementation of SARA were made by D. Berry, I. Campos, C. Carper, J. Drobman, B. Fenchel, W. Overman, R. Razouk and W. Ruggiero.
**TABLE I.---Modeling Primitives**

<table>
<thead>
<tr>
<th>TYPE</th>
<th>GRAPHICAL</th>
<th>MACHINE PROCESSABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STRUCTURAL PRIMITIVES</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A named module represents an object whose internal, fully-nested structure is hidden from the outside. A module's only possible communication with the outside is through a socket. Otherwise, a module's name is known only to the structure within which it is nested.</td>
<td>![Diagram of module A and B connected through socket C]</td>
<td>NAME_C ([A,B,C])</td>
</tr>
<tr>
<td>Example: The module, called &quot;NAME_C&quot;, is composed of modules &quot;A&quot; and &quot;B.&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A named socket is part of a module but the socket's name is known both inside and outside of its host module.</td>
<td>![Diagram of module A with socket SA, module B with socket SB, and module C with socket SC connected through interconnection NAME_C]</td>
<td>NAME_C ([A,B,C,SC])</td>
</tr>
<tr>
<td>Example: The module &quot;NAME_C&quot; is composed of modules A with socket SA, B with socket SB, and C with socket SC.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A named interconnection is a structure which connects two or more sockets.</td>
<td>![Diagram of interconnection NAME_C connecting sockets SA, SB, and SC]</td>
<td>LINK (+ [A,B,C])</td>
</tr>
<tr>
<td>Example: An interconnection called &quot;LINK&quot; connects the sockets RS, RB, and CB.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3—Structure of SARA**

From the collection of the Computer History Museum (www.computerhistory.org)
Three other behavioral modeling tools are in various stages of integration in SARA.

A simulation system called DCDS (Digital Control Design System) was implemented in 1969 and improved in the early 1970's. DCDS is not interactive but allows a designer to richly describe a system in terms of logic nets, in terms of microprogram sequences or in terms of algorithms. A fourth sublanguage called DECLARE is used to combine models described separately in the three languages: LOGIC, MICROPROGRAM and PL/I. DCDS is not integrated into the set of interactive SARA tools. Models are translated in a batch mode and simulations run in a batch mode.

The Instruction Set Processor (ISP) modeling language created by Bell and Newell has attracted widespread interest. Building upon work by Crocker a translator from ISP to PLIP was created. One complex machine model was built and is still undergoing test.

The Simulation Oriented Language (SOL) of Knuth and
### TABLE II.—Behavioral Modeling Primitives

<table>
<thead>
<tr>
<th>BEHAVIORAL PRIMITIVES</th>
<th>TYPE</th>
<th>GRAPHICAL</th>
<th>MACHINE PROCESSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>A NODE N REPRESENTS A STEP IN A PROCESS BEING MODELED. A CONTROLLED DATA PROCESSOR (SEE BELOW) MAY BE ASSOCIATED WITH A NODE TO PROVIDE INTERPRETATION OF THE PROCESS. EXAMPLE: A NODE N1 HAS A SINGLE ENTRY ARC S AND A SINGLE EXIT ARC X.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A NAMED CONTROL ARC REPRESENTS A NON-VOLATILE PRECEDENCE RELATION BETWEEN SETS OF NODES: IF THERE IS MORE THAN ONE SOURCE NODE OR DESTINATION NODE THE ARC IS CALLED COMPLEX; OTHERWISE IT IS CALLED SIMPLE. AN ENABLING TOKEN IS PLACED ON AN ARC EITHER AS A STARTING STATE OR UPON TERMINATION OF ANY OF ITS SOURCE NODES. WHEN A NODE IS INITIATED, ITS ENABLING TOKENS ARE ABSORBED. EXAMPLE: A2 AND X ARE SIMPLE CONTROL ARCS, A1 IS A COMPLEX CONTROL ARC WHOSE SOURCE SET IS NODES N1, N2 AND N4 AND WHOSE DESTINATION SET IS N5. S IS AN INCOMING COMPLEX ARC WHOSE DESTINATION SET IS N1, N2 AND N3.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Input Control Logic**

A LOGICAL RELATION AMONG THE INPUT ARCS TO A NODE SPECIFIES THE PRECEDENCE CONDITIONS THAT MUST BE SATISFIED BY TOKEN STATES FOR THE NODE TO BE INITIATED. TOKENS FROM THE INITIATING ARCS WHICH SATISFY THE INPUT RELATIONS ARE ABSORBED BY THE TOKEN MACHINE. TOKENS ARE ABSORBED FROM ONE OF AN INITIATING ARC SET GOVERNED BY AN OR RELATION IN A MANNER ESTABLISHED IN THE TOKEN MACHINE AND FROM ALL MEMBERS OF AN INITIATING ARC SET GOVERNED BY AN AND RELATION.

**Example:** If enabling tokens exist on either A1 or A2 and on either A3 or A4 then N1 can be initiated.

**Output Control Logic**

A LOGICAL RELATION AMONG THE OUTPUT ARCS SPECIFIES WHICH ARCS HAVE TOKENS PLACED UPON THEM WHEN A CONTROL NODE IS TERMINATED. WHEN AN EXCLUSIVE OR OUTPUT RELATION HOLDS, A DATA PROCESSOR INTERPRETATION MUST DECIDE WHICH ARC RECEIVES A TOKEN. WHEN AN AND RELATION HOLDS ALL OUTPUT ARCS RECEIVE TOKENS.

**Example:** When N1 terminates, its associated controlled data processor will have decided whether tokens are to be placed on B1 and B2 or on B3 and B4.

McNeely was implemented in PL1 by the Defense Communication Engineering Center under the name SOL/370. It has been installed but not fully tested and integrated. It is proposed for use as one of the SARA behavioral model analysis tools.

The design library

Every tool in the SARA system makes use of LIBRARY SYSTEM for storing and retrieving models. There are two essentially different parts of the LIBRARY SYSTEM.

- One part contains the public building block library which holds models of hardware and software subsystems that have been extensively tested or verified. These models have a great deal of value added and are assumed to be called upon strongly enough to make the investment pay off. Extensive controls on entry and modification will be exercised by a library administrator. Drobman has considered the nature of the library system with respect to hardware system design and has proposed generic building block structures for LSI based systems.

- A second part provides facilities for storage and retrieval of private building block models and private partial system designs.

LIBRARY SYSTEM is still in an early stage of development. At present we use basic facilities of the
interactive operating systems. There has been a great deal of design thinking and actual DBMS design but the results are too speculative to include here.

**Behavioral—Structural mapping**

A fundamental aspect of the SARA system is the mapping of behavior on the structural primitives i.e., modules, sockets, and interconnections. The mapping functions are currently under development. It is only after such mapping that SARA tools can detect enough cases of inconsistency and incompleteness to repay the designers for the structured sets, and interconnections. The mapping functions are cur­

**Multilevel modeling**

As indicated in the first part of this paper the strength of support for multilevel modeling procedures and the consequent management of complexity will be the principal test of SARA's reason for existence. During the partition phase, the role of SARA is to support evaluation of a refinement proposed by the designer and to accept it if the higher level abstraction is not contradicted. If the designer insists on the refinement, despite contradiction, then a procedure to weaken the abstraction is needed. Similarly in the composition phase, the role of SARA is to compose given building blocks into a model and to help discover inconsistencies with respect to the next higher level model.

The behavioral models are clearly more complex. Campos and Ruggerio have proposed formal replacement algorithms which retain equivalencies between two graph models of behavior. Whenever a single control or data graph primitive is to be refined or whenever a sub-graph can be abstracted by a single primitive, the replacement algorithm guarantees that changes are localized. In general a refine-
SL1 CODE

S2(S3,S1); S2(S3@L2,S1@L2); S2.L2+(S3@L2,S1@L2);
S3(S4,S4@L2); S3.L2(S4@L2,S3@L2);
S1(SUB1,SUB2,SUB3,SUB1@L2,SUB1@L1,SUB2@L1,SUB3@L1);
S1.L1+(SUB1@L1,SUB2@L1,SUB3@L1);
S1.L2+(SUB1@L2,SUB1@L2);

NOTE: THE "." IS USED TO DESCRIBE NESTING IN A FULLY
QUALIFIED NAME, E.G. S2.S1.SUB1 IS THE FULLY QUALIFIED
NAME OF SUB1 IN FIGURE 6.

Figure 6—Multi-level modules

In the same vein, our models of concurrent systems provide a context in which to embed verified processor interpretations and seek algorithms to prove partial correctness of a full data graph including interpretation.

For effective multilevel modeling, the "bottom line" is the accuracy, clarity and richness of the building block models available to the designer. No matter how late or early implementation decisions are explicitly made, the design process must be goal oriented and the designer must either understand the elements or have faith in the models reinforced by successful use of them. For hardware building block models, we find ourselves giving a great deal of attention to detail in order to convince ourselves that our models can accurately represent manufacturers specifications. Drobman has been able to generalize such study by defining "generic" as well as specific building blocks. There are other ways to harness rich populations of building blocks.
SL1 CODE - HIGH LEVEL MODEL

UNIVERSE (SUB1, SUB2, SUB3);
UNIVERSE (SUB1@L1, SUB2@L1, SUB3@L1);
L1 + (SUB1@L1, SUB2@L1, SUB3@L1);

SL1 CODE - REFINED MODEL

SUB1@L1 (SUB1@L1.WIRE1, SUB1@L1.WIRE2, SUB1@L1.WIRE3);
SUB2@L1 (SUB2@L1.WIRE3, SUB2@L1.WIRE2);
SUB3@L1 (SUB3@L1.WIRE2, SUB3@L1.WIRE1);
L1.WIRE1 + (SUB1@L1.WIRE1, SUB3@L1.WIRE);
L1.WIRE2 + (SUB1@L1.WIRE2, SUB2@L1.WIRE2, SUB3@L1.WIRE2);
L1.WIRE3 + (SUB1@L1.WIRE3, SUB2@L1.WIRE3);

Figure 7—Multi-level interconnection

For example we have a path for translating ISP descriptions into PLIP interpretations but we do not yet fully understand limitations imposed by efficiency issues.

In the companion paper, a software example is taken all the way from programming-in-the-large to programming-in-the-small or PLI code. In that case the PLI language definition provides the building blocks and the transition is smooth because the entire SARA System is written in PLI. When concurrency exists, the transition is not completely smooth because the "token machine" is an integral part of the SARA environment and there must be equivalent machinery wherever a SARA model is expected to execute. In fact we are exploring new architectures in which SARA models would run with no alteration of the designed and tested model.

SUMMARY AND CONCLUSION

The UCLA design methodology has been elaborated in an attempt to indicate its requirement-driven and self-documenting properties for modular, concurrent, hardware and software systems. The need for implementation in hardware, for example, may show up explicitly in initialization or finally appear in composition.

Those goal characteristics which distinguish this design philosophy are:

- Explicit modeling of environments using the same primitives as the systems being designed.
- Separate but related static and dynamic models.
- Separate but related control and data flow models.
Separate but related structural and behavioral models.
Support for both top-down and bottom-up design.
Implementation decisions forced in composition but retained when they are entered earlier.
Supported "well-behaved" properties in multilevel modeling and analysis.
Supported simulation of concurrent processes.
Supported fabrication.

SARA is useful now as a collection of tools which help structured design of concurrent systems. SARA has been tested in a number of design problems. As a result of its recent move to MIT-MULTICS, it is soon to be tested on larger system problems. To the extent that support of the requirement-driven multilevel design methodology is increased and detected "errors" are reported to the designer, SARA's value will increase significantly. A number of problems with the current methodology and its support are discussed in the companion paper. 5

We believe that SARA is unique in the support it gives to design of concurrent systems. We do not look on this methodology as being in competition with other active work in, for example, dataflow architecture or sequential software design. We are constantly alert to incorporation of other maturing synthesis techniques, particularly when any of our designs reach the stage of incorporating sequential programs.

The number of poorly understood problems which have been pointed to in this paper is very large. They give credence to the author's philosophy which rejects completely-automatic design at this time and seeks to support the designers injection of value judgments and decisions. Automated design procedures should be introduced selectively and patiently. Sarah begat Isaac when she was ninety years old [Genesis]. It is possible that, with the help of SARA and the help and the wisdom of many researchers and machines, we will know enough before 2067 to beget a powerful ISAAC (Information System Automatic Architect Computer). We do know how to beget an ISAAC but also know that a 1978 ISAAC would be fired for incompetence long before tenure could be achieved.

REFERENCES