A SYSTEM THROUGH HARDWARE DESIGN METHODOLOGY

Larry D Anderson and Earl M Vraa
Defense Systems Division, St. Paul, MN

1. ABSTRACT

Sperry's entry into the Department of Defense's (DoD) Very High Speed Integrated Circuit (VHSIC) program enhanced our understanding of at least two issues important in Sperry's business planning. The issues include 1) the dangers of complacency with respect to technology advancements and 2) our strengths with respect to the development of cost and technically competitive electronic products. The former was dramatically illustrated when Sperry's proposal for the development of a VHSIC military standard computer brass board was rejected for brassboards encompassing entire subsystem capabilities including the computer. On the positive side, Sperry's proposal for a Computer Aided Design (CAD) system was selected and a subcontract awarded. During the contract, our proposal objectives have become a model for subsequent work by both the government and other members of industry.

The understanding derived from the VHSIC program can be summarized simply. Sperry's future DoD system business will inevitably require us. to design (or adopt existing designs) and package totally integrated digital hardware components on a chip, e.g., wafer scale integration. In the opinion of DoD and other members of industry, this will require a hierarchical CAD system and design methodology. This paper will define the concept of hierarchical design. The concept of hierarchical design shall be illustrated as it impacts system designers, hardware logic designers, software designers, and CAD support system personnel. Combined with requirement specification and correlation, the concept provides a framework for enforcing a system-through-hardware design methodology.

2. TECHNOLOGY EVOLUTION IMPACTS CAD

The DoD initiated the Very High Speed Integrated Circuits program in 1979 to meet presently defined and future military system needs. The objective was to provide increased ability, on the part of the U.S. electronics industry, to respond fully and quickly to the DoD's continuing and rapidly expanding requirements for complex, high-speed signal processing functions in its electronic subsystems and systems.

As part of the VHSIC Phase 1 proposal solicitation [1], DoD requested a "hierarchical chip design methodology for use with computer aided design (CAD) facilities." The CAD tools were to "support the entire design process including functional definition. Design, implementation, functional verification (and) physical verification." The CAD tools were to be "integrated with a unitized data base into which data items need be entered only once and are then available to all tools and aids within the CAD system."

Since the initiation of the VHSIC program, the concept of hierarchical design has received a considerable amount of emphasis from both industry and the DoD. For example,

Mr. John Hanne, vice president and CAD project manager for the Microelectronics Computer Technology Corporation (MCC), Austin, Texas, recently toured the country describing a model for their CAD system R&D. The model contains a hierarchical description language and data base which, when combined, are capable of supporting hardware and firmware descriptions for CAD tools [2]. In the Commerce Business Daily [3], the DoD announced the first phase of the Integrated Design Automation System (IDAS-I) program. "The purpose of IDAS-l is to support VHSIC Phase 2 brassboard designs (and) will address all levels of electronic system design from system level through chip level."

Discussions between the authors and several members of industry (e.g., RCA, Hughes, Westinghouse, TRW, TI, etc.) indicate *all* have some level of activity underway to better understand the concept of hierarchical design and its impact on new and existing CAD tools. It is this subject which shall be discussed and illustrated in the remainder of this paper. Concepts and illustration ideas have been collected from previously published work in the public domain.

3. HIERARCIHCAL DESIGN ILLUSTRATED

At this point the concept of hierarchical design is just that - a concept, an abstraction. It has meaning only in the confines of an individual's experience. The vast majority of designers (including system or circuit; software or hardware) neither appreciates its potential nor denies its benefits.

To illustrate, consider a recent presentation given at Sperry Defense System Division (DSD) on the utility of the hardware description language ULYSSES [4]. The presentation was given by a logic designer who recognized the requirement and potential for a language capable of describing hardware design and subsequently verifying that the described functionality (behavior) was correct.

The presentation covered the utilization of ULYSSES in the design of a 24-bit adder for the

```
1   B.ADDN/TBMPL
2   macro addn: :-'ADDN'
3
4   templ     ADDN  =  <<pars(sig a b)
5
6         int n = size(1,a)
7
8         repl [0…n-1]  a   b        sig cy out
9
10        def out cy = add1bit(a b cy[1… •• n-l]&qnd
11
12  out cy[0] >>
13
14  */
15  */
16  */  This template defines an N-bit adder
17  */
18  */  the following template is called directly from this template
19  */                    ADDIBIT
20  */
```

Figure 1. N-Bit Adder Defined Using One-Bit Adders

```
1   B .ADD1BIT/TEMP
2   macro add1bit: : 'ADDIBIT'
3   temp1 ADD1BIT = << pars(sig a b cin)
4
5   (a xor b xor cin) , {(a and b or (a and cin) or (b and cin) >>
6
7   */
8   */ This template describes a l-bit adder function
9   */
10  */
```

Figure 2. One-Bit Adder Defined Using "Hierarchical" Capabilities of ULYSSES

Sperry Micro 1100 processor. The designer chose to describe the adder generically as an n-bit adder (Figure 1) which subsequently achieves the proper functionality by utilizing a one-bit adder description (line 10, Figure 1). The one-bit adder description is shown in Figure 2. The data types (signals) and basic logical operators (XOR, AND, etc.) available in the language are illustrated in lines 3 and 5 (Figure 2) respectively.

To the designer, the language was hierarchical because in describing the n-bit adder it was possible to invoke the one-bit adder by name without describing the detailed logical. The capabilities of the one-bit adder are made: available using the TEMPLate (or library) capabilities of the language (lines 1 and 3 of Figure 2). Similarly, the n-bit adder could be invoked by other designers without knowledge of its implementation logic since it had also been described using the TEMPLate capabilities (lines 1 and 4 of Figure 1).

The problem with interpreting the term hierarchical in this way can be made visible by observing what occurs when the language is translated (compiled) into a form by which it can be used for simulation. In this example, several machine instructions were needed to simulate the one-bit adder. These instructions are necessarily replicated or iteratively invoked to complete a 24-bit add. During the presentation, a question was asked about the amount of time required to simulate a single test vector with 12-bit operands. The answer: more than one second of Univac 1100 CPU time. Multiply this by the complexity of tomorrow's chips and the number of test vectors necessary· to exhaustively test a design and the magnitude of the problem becomes apparent.

The solution to the problem requires that the design language be capable of expressing the functionality for CAD tools (e.g., a simulator) in more generic or abstract terms. For example, Figure 3 shows the logic schematic for a two-bit adder shown with carry truncation to simplify the illustration. Expressed in ULYSSES (with current operators and data types), the functionality must be expressed as four logical operations. When simulating the hardware functionality, the passing of signals between the logical operators also utilizes machine instructions. Alternatively, Figure 4 shows precisely the same functionality, i.e., add the information on Rl to the information of R2, placing the results on R3. The hierarchical capabilities of the language represented here differ significantly from those available in ULYSSES. The functionality expressed by the more abstract form can generally be emulated with as few as two or three machine instructions during simulation, thus taking only microseconds to complete". Further, this addition is possible not only for the two-bit adder but also for the earlier discussed 24-bit adder that took more than a second to simulate.

In fairness to both the designer and the ULYSSES language developers, it must be noted that extensions to the ULYSSES data types and basic operators are planned. However, they were not then and, to the best of our knowledge, are not now in place. Several people at the presentation took this definition of hierarchical as an ultimate definition. To extend the hierarchical capabilities of the language and associated language processor to their fullest potential, hence quality as hierarchical is expected to take several man-years of labor.

4. HIERARCHICAL DESIGN AUTOMATION - HARDWARE

The functional equivalence represented by Figures 3 and 4 illustrate several requirements imposed on hierarchical design languages and associated design automation tools. Three requirements and their impact on design automation tools are discussed. First, Figures 3 and 4 illustrate, respectively, two distinct capabilities needed in a hierarchical language: 1) a means to express functionality (behavior) for each node (or partition) in a design; and 2) a means to describe the interconnection (structure) of nodes. (If the language allows reference only to functions that are predefined, i.e., uses models that are built into the CAD tools, the language is primarily a structural language. The language generally supplies only interconnect information usually referred to as "net lists." A large percentage of existing CAD systems use structural languages thus cannot be classified as hierarchical. The majority of CAD systems in use by Sperry fit this description.)

Second, a hierarchical language must support user-defined data types. For example, in Figure 5 (Figure 3 redrawn), the associated language need only process "signals" which propagate between gate-level operations or primitives. While the signals may take on several "states" (e.g., high, low, undefined, high impedance, etc.), the "data type" is predefined by the language to reduce the complexity of the language compiler. The need for user-defined types was illustrated in both Figure 3 and 4. Here, ports into or out of the adder component are "variables" that must be defined to hold a range of integer values from 0 to 3. A truly hierarchical language must support a wide variety of types including enumeration (e.g., for signal states) as well as bit, integer, reals, floating points, etc.

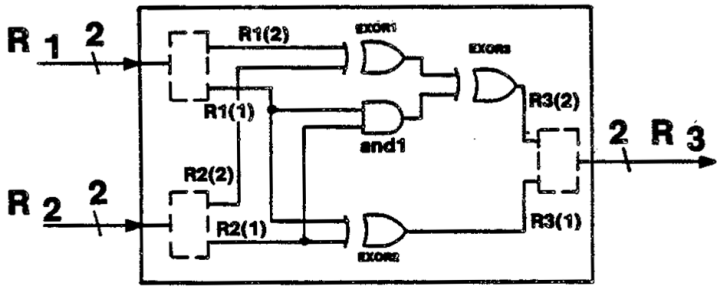

Figure 3. 2–Bit Adder with Carry Truncation

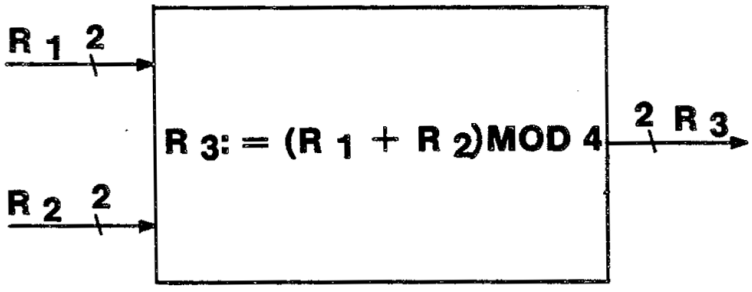

Figure 4. Functionally Equivalent 2–Bit Adder

Third, a hierarchical language together with CAD tools must support the substitution of detailed designs or structures (see Figure 3) for less detailed but functionally equivalent designs (see Figure 4). This requires the use of data conversion (also referred to as coercion) functions, when performing the substitution. The concept of coercion functions is represented in Figure 3 by the "dotted boxes." The functions are needed, for example, when substituting the hierarchical representation of Figure 4 with the more detailed representation of Figure 5. In a designer friendly CAD system, the coercion functions are implicitly inserted by

the tools, however, their existence (build-in and/or user-defined) must be supported by the language.

The requirements just discussed help establish a framework around which hierarchical design tools can be developed. For example, a simulation generation system can now be developed which uses structural descriptions to collect both high-level and/or primitive-level behavioral descriptions and form an efficient simulator [5]. To illustrate, consider the simulator shown in Figure 6. The simulator consists of 1) a collection of hardware behavioral (models); 2) routines for interpreting commands acceptable to the simulator (user interface); 3) routines for connecting models or instances of models (instantiation driver); and 4) routines which extract data from the models during simulation (monitors). The latter provides the framework for built-in simulator command implementation and will support user –defined routines (e.g., routines capable of printing timing diagrams, recording test probe results, etc.).

The ability of the collected software to exchange data and operate concurrently to simulate hardware designs is provided by the control kernel. The control kernel also activates "coercion functions" as needed to convert data between different levels of design. The identification of models or monitors, the number of times each model or monitor is used in a design, and how the model or monitor "instances" are interconnected is established by the instantiation driver from the structural description.

The importance of structural and behavioral descriptions on CAD tools can be seen from the design illustrated in Figure 7, i.e., a hierarchical design for a Digital Adaptive Voting Ensemble (DAVE). Using an interactive tool (later shown as a hierarchy processor) to select a desired design from a hierarchical design data base, it is possible to replace high-level behavioral descriptions (functional, block, register transfer level, etc.) with a structure or structures of lower level (more primitive) behaviors. This enables designers to utilize earlier, more abstract, designs, and simultaneously concentrate on specific areas of interest (i.e., implementation detail) while using CAD tools which operate efficiently.

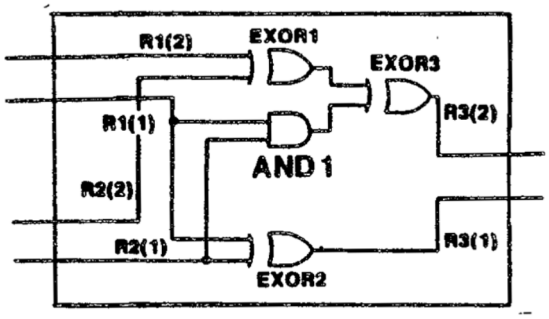

Figure 5. A Hardware Design Model Accepting Only Signals

The hierarchical use of structure and behavior for simulation is useful in an analogous manner for automated hierarchical layout. During layout, a behavior can be viewed as a "composite cell" having physical size attributes. The values of higher level composite cells (PLAs, ROMs, ALUs, etc.) are determined by the structure (interconnect and routing) requirements of lower level composite cells (polygons, transistors, gates, etc.).

In general, languages which have not separated behavioral descriptions from struetura1 descriptions have stymied the development of CAD tools which support hierarchical design. Clearly, the separation of structure and behavior within a hierarchical hardware description

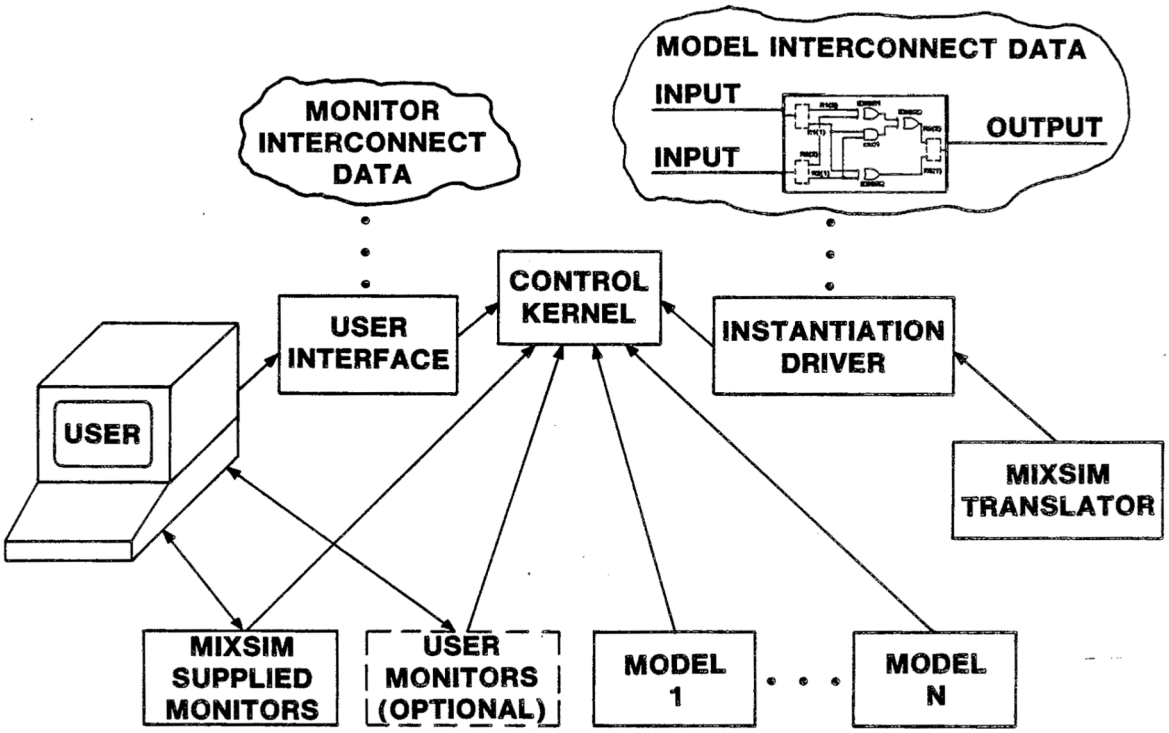

Figure 6. Program Structure Resulting From Simulation Generation

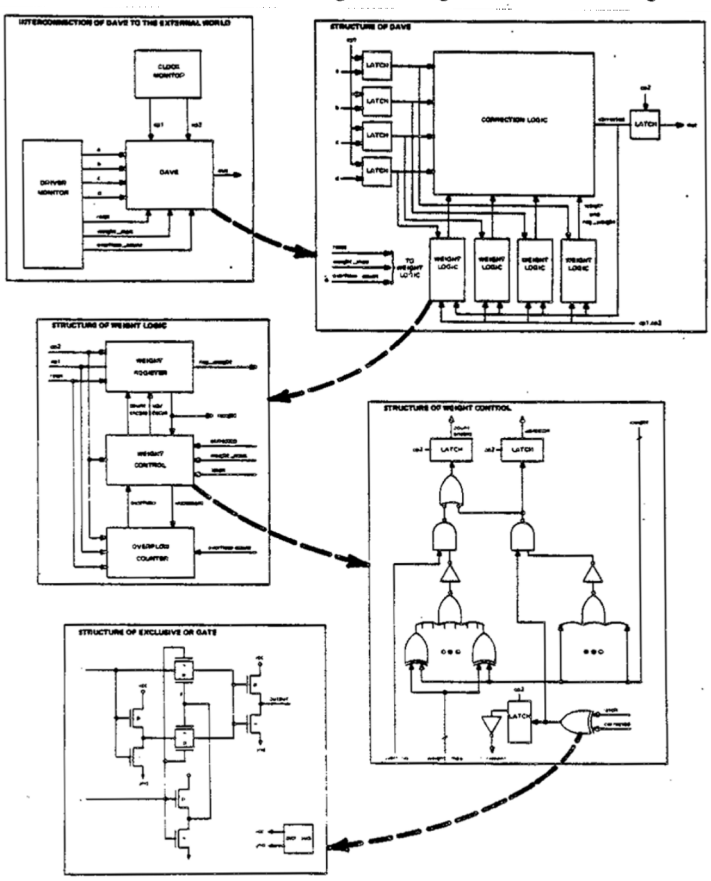

Figure 7. DAVE Illustrated with Hierarchical Structures

language provides a consistent framework for resolving complex technical problems in tool development, i.e., data type; conversion, access to physical attributes of composite cells, establishing artwork boundaries, and supporting logic partitioning. Separation also simplifies communication between designers, particularly across levels of design detail.

5. HIERARCHICAL DESIGN AUTOMATION - SYSTEM

At this point in our paper we have illustrated the concept of hierarchical design as applied to hardware or chip design. The use of hierarchical design was accelerated by the DoD VHSIC program as hardware technology advanced to the point where entire digital subsystems could be placed on a single chip. In general, it has forced

industry to look beyond traditional tools used for hardware design to capabilities previously referred to as system design tools.

Sperry DSD has developed and is continuing to refine a CAD system model (Figure 8) which addresses system-through-chip design. Shaded portions of the system model have been designed and are under development or have already been completed.

The language used for expressing design was named the Hierarchical System Language (HSL) [6]. Ada* syntax and semantics were chosen as a base for the language in order that the most powerful means for expressing machine readable functionality would be available for design expression and associated verification To this base were added capabilities by which timing, concurrency, structure, physical descriptions, etc. could easily be described .by the hardware designer.

Note: *Ada is a registered trademark of the Department of Defense

The language is processed (compiled) and stored in a Hierarchical System. Data Base (HSDB) in a form readily accessible by design automation tools. The simulation generation system discussed earlier together with a runtime simulator completes a core CAD model system around which specific tools required for all levels of design can be integrated. The remainder of this paper shall describe and illustrate how this core tool set is required by and can be used for system through chip design.

The design process can be thought of as a series of steps that for the most part are universally recognized and accepted. For example, the following steps are equally well understood by system designers and chip designers.

1. Requirement Definition - Identification of specific objectives and capabilities necessary to solve a problem.

2. Design Hypothesis - Conceptualization of candidate approaches/functions needed to attain required objectives.

3. Design /Validation - Verification that interrelationships, behavior, and characteristics of conceptual design meet the required objectives.

4. Design Synthesis - Selection or modification of individual functions or parameters such that the physical implementation is the best possible in terms of required needs.

5. Design Implementation - Development of the final design using physical elements defined by the synthesis process.

Figures 9a and 9b illustrate two of several possible levels in hierarchical design - subsystem and microcell design. The subsystem design begins with requirements (not shown) translated into activities (tasks or functions) that must be performed (left half of Figure 9a). The designer is immediately faced with the issue of performance, i.e., which activities must be performed in

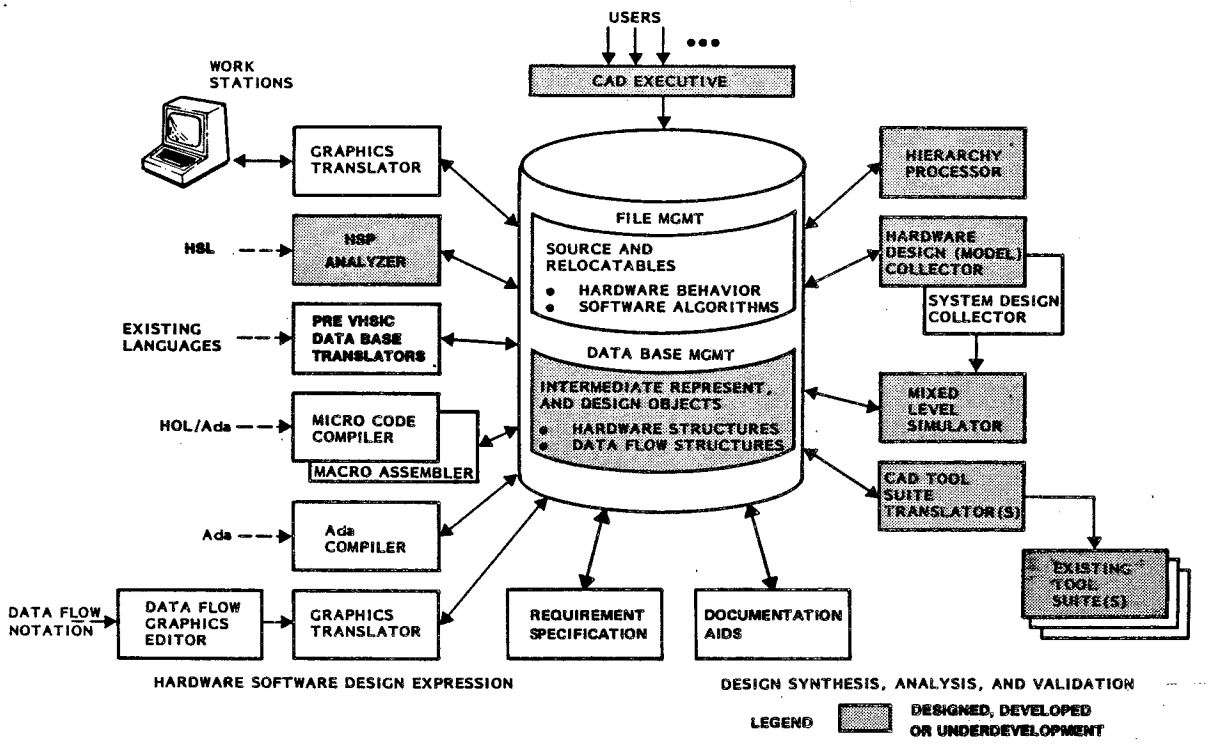

Figure 8. A Model for a Hierarchical CAD System

parallel or, equivalently, which activities will result in performance bottlenecks.

At this stage of design, an experienced designer resorts to analytical methods of design verification. Using empirical data collected from previous projects, an analysis of data flow and functional partitioning is performed. Notice, however, that the system designer, like the hardware designer, is interested in both behavior and structure of nodes. Whether a system designer must proceed with detailed functional implementation design (and associated cost) or can map to existing physical entities is dependent upon a number of less tangible factors including the results of analysis, and the criticality of requirements. Nevertheless, the authors of this paper believe that the ability to describe and simulate functional designs, analytical designs or a mixture within a single CAD system is important. The potential for doing so, using the core tool set, is presented in the final section of this paper.

At some point the designer must map a satisfactory logical representation of a design to a physically realizable entity or col1ection of entities (shown in the right side of Figure 9a as a set of processing elements). If all physical entities exist (i.e., have been designed, implemented, and are available for integration/test), a comparison of satisfactory test results against requirements completes the designer's work. If one or more physical entities do not exist, the designer returns to the logical level of design for that entity or set of entities. It should be noted, however, that premature mapping using prototype development, integration, and test is costly.

The process of mapping the logical (i.e., functional) to the physical must always be performed even if delayed until the lowest level of design, herein represented by the microcell design in Figure 9b. Logical to physical mapping can occur at any design level. In the pre-VHSIC era for example, logic level designs were mapped onto commercially available MSI/LSI chips. However, like premature mapping, prolonged delay in mapping logical to physical (i.e., failure to use bottom-up designs) can also be costly. As a result, hierarchical CAD tools are needed to verify the mapping feasibility at any level. This requires that behavioral models of both logical and previously developed (i.e., physical) designs be retained in the data base.

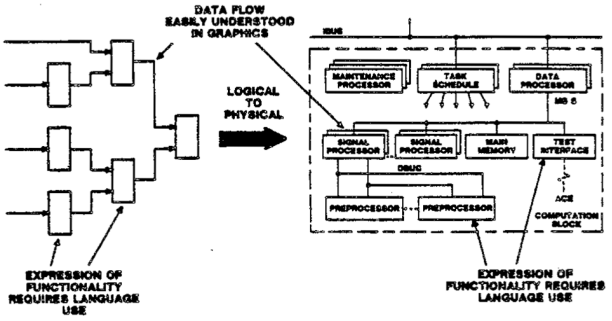

Figure 9a. Subsystem Design

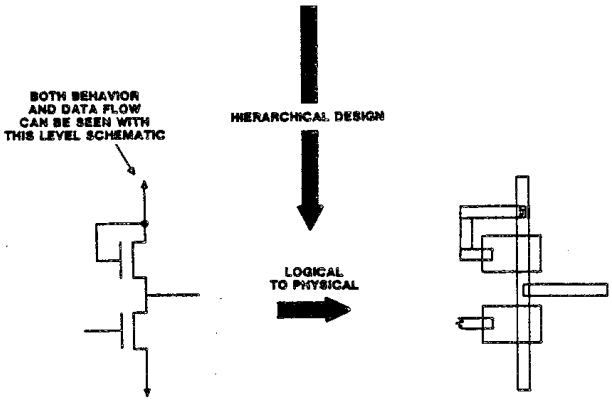

Figure 9b. Microcell Design

Examples of the range of logical design levels between 9a and 9b were shown in Figure 7. In a hierarchical design system, each logical design brings requirements closer to a realizable physical implementation. Each level can be automatically compared to the previous level and the design corrected before the more costly physical mapping and integration occurs.

6. ANALYTICAL VERSUS FUNCTIONAL DESIGN VERIFICATION

In the previous section it was observed that design often involves analytical rather than functional modeling. For example, at the circuit level an analysis of capacitance, current loads, or thermal characteristics is required. At the system level, an analysis of queue lengths and associated timing delays is performed, often using values statistically derived from empirical data.

A hierarchical CAD system must support both analytical and functional design verification. A major contributor is the use of separate structures and behaviors combined with the standardization of internal data base representations. To demonstrate the potential of this contribution, refer again to the data flow graph shown on the left side of Figure 9a. The illustration represents a structure of concurrently executing nodes or behaviors.

Typically a system designer would analyze this structure to determine which of the concurrent node behaviors must run in parallel and which can be executed sequentially while meeting performance requirements. Often, a designer uses a Petri-net model to formalize the concept of data traversal through the structure [7]. The model provides the designer with the means of assessing the arrival rates of data (referred to as tokens) at nodes, the impact on computational rates of the nodes, the backup of tokens at each node (queue size) and the potential of tokens

```
1.      MODULE box (inl, in2, output) IS
2.          TERMINAL inl, in2: IN arc_type;
3.          TERMINAL output: OUTPUT arc_type;
4.
5.              --PARAMETER DECLARATIONS, E.G.
6.                -- MAXIMUM QUEUE SIZE (maxsize)
7.                -- TIMING DISTRIBUTION (distr_id) FROM EMPIRICAL DATA
8.
9.              PARAMETER delayed:   real;          = 1.0;
10.             FOR output 'delay' USE delayed;
11.     END MODULE box;
```

Figure l0a. Analytical Model Declaration

```
1.      MODULE BODY box (inl, in2, output) IS
2.      BEHAVIOR abc IS
3.
4.          -- LOCAL DECLARATIONS, E.G. queue, last_delay
5.
6.      BEGIN
7.          -- TERMINAL LINKAGE IMPLICITY PERFORMED BY SIMULATOR FROM
STRUCTURE
8.          -- PARAMETERS POPULATED FROM DATABASE OR REQUESTS TO DESIGNERS,
E.G.
9.              -- MAXIMUM QUEUE SIZE (maxsize)
10.             -- TIMING DISTRIBUTION DERIVED FROM EMPIRICAL DATA (distr_id)
11.
12.     LOOP
13.         IF SIM_TIME < = last_delay AND tokens_present (in1, i02) THEN
14.             consume_tokens (in1, in2);
15.             produce_token (output);
16.             last delay + time_distr (distr_id);
17.         ELSE
18.             increment queue (in1, in2); _ _ FOR TOKENS PRESENT
19.             IF queue (in1) OR queue (in2) > = maxsize THEN
20.             inform_designer (maxsize);
21.             ENDIF;
22.         ENDIF;
23.     END LOOP;
24.     END MODULE;
```

never traversing through the structure (a form of deadlock).

Notice that it is not necessary to actually represent the functionality of the node, but rather useful data can be derived using only statistical representations of time and knowledge of the availability of space for data queues. The authors believe that this analytical simulation can be performed using the core CAD model simulation generation system and hierarchical system language. To illustrate, consider the model description (MODULE Description and MODULE BODY) in Figure l0a and l0b. This description can be instantiated to represent each node in, for example, the structure of Figure 9a. While the model was written for only two inputs to simplify the illustration, it could easily be expanded for "n" terminals (ports) and added analytical capability. The model requests attributes from the data base implicitly or can be written to request attributes explicitly from the designer (lines 9 and 10, Figure l0b) using the PARAMETER capabilities (lines 5, 6, and 7, Figure l0a) of the language.

The model is implicitly activated as each token arrives. If all necessary tokens are present and processing time available (line 13, Figure l0b) the token will be processed (consumed – line 14). Subsequently, an output token or tokens will be generated (line 15) and the time used in processing computed (line 16). If tokens cannot be processed; they are queued (line 18). Queue sizes (or queue overflow) are passed (line 20) to the designer.

The submission of a structure (see: Figure 9a) and the analytically oriented behavior (Figure 10) to a hierarchically oriented simulation generation system enables an efficient runtime simulator to be constructed. Using this technique, the definition of functional models for selected nodes would also enable an efficient runtime simulator containing a mixture of analytical and functional models to be generated. Theoretically the simulator could use analytical behaviors for pre-assigned physical components (i.e., physical components that are known to meet performance criteria) and functional behaviors for that part of a system design where technology infusion is required. The authors contend that these concepts of hierarchical design and associated tools arc needed to advance design automation and synthesis.

7. ACKNOWLEDGEMENTS

The concept of hierarchical design as supported by CAD tools is complex and will continue to evolve. In this paper, the authors have attempted to illustrate what we see: occurring. Many of the ideas expressed have been borrowed from those people with whom we have been privileged to work including Mr. Mark D. Glewwe, Mr. Allan W. Kiecker, Mr. Dean R. Kioker, Mr. Steve J. Piatt, and Mr. John J. Travalent. Thank you for your contributions.

8. REFERENCES

1. “Information to Offerors or Quoters," Contracts Branch, FMCO, USAERADCOM, Ft. Monmouth, New Jersey, September, 1980.

2. Hanne, J. R., “The VLSI/CAD Program, “ Microelectronics Computer Technology Corporation (MCC), MCC Member Confidential, January 1984.

3. Hines, Dr. J., "Integrated Design Automation System," Commerce Business Daily, Issue No. PSA-8504, Wednesday, January 18, 1984.

4. "ULYSSES Language - Preliminary User Guide and Syntax Definition," Sperry Micro Products Development, April, 1983.

5. Kiecker, A. W. and Glewwe, M. D., "A Simulator for Hardware Models at Multiple Levels of Design Detail," Proceedings of the 21st Annual Spring Technical Symposium, Sperry Corporation, Volume 1, May, 1983.

6. Piatz, S. J., "HSL - A Comprehensive Hardware Description Language," International Conference on System Sciences, Volume I, 1984.

7. Frank, G. A., "ADAS - An Architecture Design and Assessment System," Research Triangle Institute Documentation, 1984.