The ADAM Advanced Design Automation System: Overview, Planner and Natural Language Interface

John Granacki, David Knapp and Alice Parker

University of Southern California

Abstract

This paper describes ADAM, an integrated Advanced Design AutoMation system, with focus on the knowledge-based synthesis subsystem. Working parts of this subsystem include a number of design activities and utilities, and a unified, multidimensional, hierarchical design representation. Two aspects of the synthesis subsystem are described in detail: the design planner and the natural language interface. The planner builds a plan for synthesis and analysis activities, drawing inferences from a knowledge base represented by a semantic net. The natural language interface accepts system-level behavioral specifications. Both of these packages are currently being implemented.

1. Introduction

This paper reports on an integrated system, the ADAM (Advanced Design AutoMation) System. ADAM is a unified framework with a restricted natural language interface, a database for the design representation and knowledge base, and a knowledge based planner which supervises specialized procedures and rule-based systems for synthesis, analysis, and custom layout.

The major subsystems of ADAM are custom layout tools, an expert system for the design of testable circuits [2], and a knowledge-based synthesis subsystem, which is the focus of this paper.

Custom layout includes optimal power-ground routing [3] and a CMOS silicon compiler. Some modules of the silicon compiler have been tested. The compiler determines the dynamic placement of transistors within cells and lays out the cells. The layouts will adhere, where possible, to timing constraints.

2. Knowledge-Based Synthesis

The synthesis subsystem is shown in Figure 2-1. The goals of this subsystem are to produce correct implementations, produce a range of implementations with varying tradeoffs, allow the amount of designer interaction to vary, start with a partial design in place, design incrementally, and design according to constraints. This system contains a **planner**, which examines a specification of the design contained in the **design representation** and knowledge about the design process (contained in the **knowledge base**), and builds a design **plan**. **Design activities** and **utilities** operate on the design representation, using the knowledge base specified in the plan. PALLADIO [4] inspired this work.

2.1. Design Activities

Supported design activities include synthesis and design verification. A package which synthesizes clocking schemes has been completed [8], and a general data-path synthesis framework has been prototyped [9]. This framework has shown that synthesis and verification can share a common methodology. Utilities which perform checks for specific classes of design errors are being written. The completed V collision detector [10] is guaranteed to find potential value and operation collisions in registers, buses, and operators.

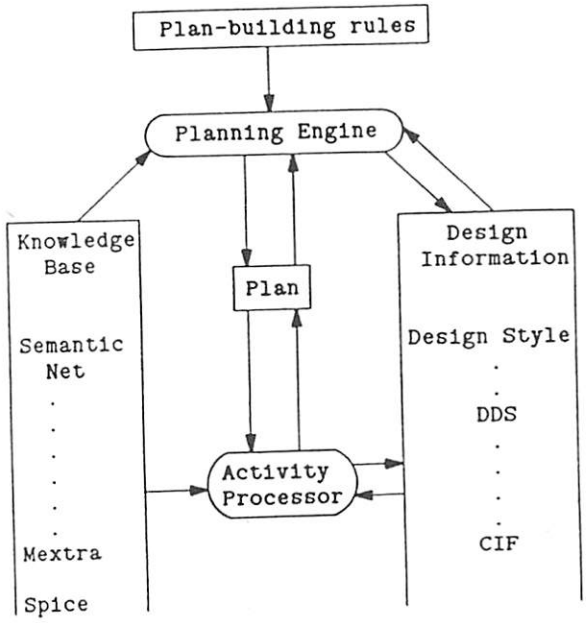


Figure 2-1: Overall Organization of the Synthesis Subsystem

The design activities and the planner rely on general-purpose utilities, including a working critical path finder and estimators. The area estimator, ARREST, provides estimates of area and aspect ratio for standard-cell circuits. One result of this project is the discovery of a strong correlation between the number of switches (driver transistors) and cell area for NMOS and CMOS cells. Another result validates Rent's rule [7] for polycell layouts, and describes the circumstances under which the rule holds [6].

2.2. The Semantic Database

The design representation and design knowledge are stored in a semantic **database** [1]. All information is represented as objects. The design activities, utilities, and the planner all operate by manipulating knowledge-base and design objects. A schema for design information exists and is being extended to incorporate design knowledge. The fundamental design object is the **component**, which has attributes that describe it in terms of a **dataflow model**, a **structural model**, a **timing model**, a **physical model**, a set of **bindings**, and information about the component's status.

Funding for this paper was provided by NSF through Computer Engineering Grant # ECS-8310774.

The component object type is a representation of a system or subsystem, in which the models are unified and interrelated by interspace bindings. Components are used to represent both packaged subsystems and the target design; components might represent a 7400 TTL part and a complete mainframe. A collection of components forms a component library. Also implemented for ADAM is the 3DIS (3 Dimensional Information Space) [1], a way of representing data to the user so that browsing through the database is easy and efficient.

2.3. The DDS Design Data Structure

Within the design objects, design data is modeled with the Design Data Structure (DDS) [5]. The DDS is a multilevel design representation. It partitions design information into "subspaces" so that there are no implicit relationships between the objects of one subspace and the objects of another. The structural, timing, behavior and physical characteristics of a design do not decompose into isomorphic hierarchies. Thus, the representation uses separate hierarchies.

- The **data flow behavior subspace** specifies behavior by means of a single-assignment language. It is internally represented like a data flow graph except that arcs and nodes must be subscripted if they belong to loops, and it is hierarchical.

- The **structural subspace** is analogous to a schematic diagram. It is a recursively defined hierarchy of "modules", which represent logical structures, and "carriers", which contain values.

- The **physical subspace** contains physical properties. Its primitive elements are "blocks" and "nets", and contains attributes such as power dissipation and size. Layout information is contained in this subspace.

- The **timing and control subspace** is a hierarchy of time "ranges", each of which is a partially ordered set of "points". The timing subspace encompasses timing diagrams, timing constraints, and control-flow graphs.

The four subspaces are algebraically related to one another by four types of interspace bindings. No other interspace relations, exist, even implicitly.

1. **value to carrier to range**: this three-way binding states that a given value is to be found on a given carrier (i.e. a structural element capable of carrying a value) over the given time range.

2. **node to module to range**: this binding states that a behavioral node is implemented by a particular module, and that the module is performing that function over the given time range.

3. **block to module**: this binds a block of the physical subspace to a module of the structural subspace.

4. **carrier to net**: this binds a carrier to a physical conductor, for example a data bus.

Figure 2-2 shows an example of the subspaces and bindings, illustrating the non-isomorphism of the four hierarchies. The bindings are shown by arcs connecting the hierarchies. The arc connecting the add in the data flow indicates the time range when the add is performed and that the ALU performs the add. A graphical interface to the DDS, AGIS, has been completed.

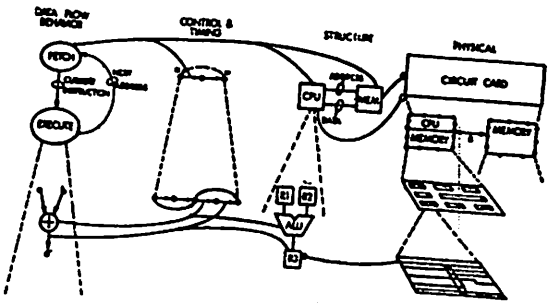


Figure 2-2: The Four Design Subspaces

3. The Design Planner

The design planner aids the designer in interactively selecting synthesis and analysis tasks, in determining which design techniques to use, and in setting up and monitoring design constraints. A number of accomplishments can be reported on here. These include

1. the classification of design tasks into types, the enumeration of design techniques, and the determination of commonly used analysis routines,

2. a structural framework for the planner,

3. the design of a knowledge base for design tasks, design techniques, and design alternatives,

4. the design of a unified data structure for the representation of history, current status, and future plans,

5. the selection of measures to be used by the planner in determining ordering of tasks and selection of design alternatives, and

6. a method for construction of plans using the measures and knowledge representation.

3.1. Framework of the Planner

The planner includes a knowledge base, which contains the taxonomic and methodological knowledge. Second, there is an abstract representation of the current system state. Third, there is a representation of past and future states, i.e. of history and plans. Fourth, there is a planning engine that builds plans from the knowledge base. Fifth, there is a mechanism for maintaining 'demons', (e.g. area budget enforcement demons) which monitor the design process and cause re-evaluation and recasting of plans. Finally, there are support components used to update and derive the abstractions that constitute the planner's 'state', and to mediate the implementation of its decisions.

3.2. Knowledge Representation

The knowledge base is a set of interpenetrating graphs, similar to semantic nets and context models [11]. It contains knowledge including taxonomy (e.g. ' a computer consists of CPU and memory...'), methodology (e.g. 'One can transform a serial design to a parallel one by...'), and other information, (e.g. the performance benefits of a cache). An example fragment is shown in Fig. 3-1.

3.3. Representation of Design Time: History and Plans

Plans and histories alike are represented as annotated tours through the semantic graph; in a history the annotations represent achievements, whereas in a plan the tour is hypothetical and the

annotations represent goals and objectives. The planner constructs these tours.

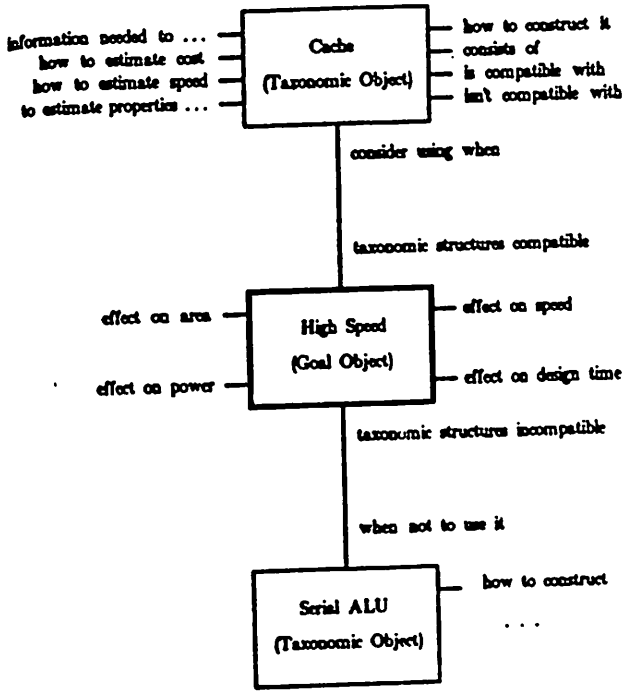


Figure 3-1: Fragment of the Knowledge Base

3.4. Measures Used by the Planner

Two types of measures are used by the planner to prune the design space, those which describe attributes of designs, and those which describe attributes of plans and techniques. Estimates of design attributes vary in accuracy, depending on the estimation technique and the state of the design. Attributes of plans and techniques, such as run-time and quality of results, are usually estimated with a fixed technique, e.g. where a method runs in $O(n^2)$ time.

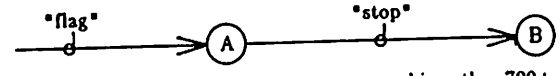
Attributes of the design include its taxonomic classification, its area, speed, and power consumption, and (recursively) the attributes of its subunits. For example, a particular cache memory might be taxonomically classified as **write-through**, in addition to the physical attributes of **power** and **area**, and the timing-space attribute of **cycle time**. Caches in general have attributes of going well with high speed design; of being particularly useful where processor and memory speeds are mismatched; of increasing costs; and creating data-dependent slowdowns (i.e. misses).

Attributes of plans and techniques include expected plan/technique run-time, the **probability** of successful application, the **utility** of a successful application (e.g. it will provide 50 % performance gain over other techniques), and an estimated **reliability** of the probability and utility estimates themselves. The measures are all context sensitive, depending on the current design. As an example, consider the technique "Allocate using Mixed Integer-Linear Programming" where the problem size is large. The probability of success is high, but the utility is low because of the computational complexity. The reliability of the estimated probability and utility measures is high because the algorithm's properties are well known.

4. Natural Language Interface

A natural language interface for entering system specifications is being developed. This will facilitate the construction of a complete, correct, consistent, concise and comprehensible specification, via an interactive dialogue with the designer. The interface will allow the user to assist in the disambiguation of the input, simplifying a difficult part of natural language processing. One component of the interface is PHRAN, a phrasal analysis program [11] which accepts English sentences and captures the semantic content in a declarative representation using concept-pattern pairs. The subsequent representation of the sentences will be mapped by SPAN, a specification analysis program, into the DDS representation.

For example,
Specification Text: On receipt of a flag, activity A sends a command, stop to activity B.
Pattern: [(condition)(s-activity)(root send)(value) (to(r-activity))]
Data Flow portion of concept in DDS:



Analysis is currently being done to combine the 700+ word vocabulary into larger phrasal units and to find recurring language structures in specification texts.

The list below shows a small fragment of the vocabulary.

acknowledge	bus	concurrent	multiplexer
address	byte	device	receive
asynchronous	command	event	send
block	communicate	handshake	transfer

List 1. Vocabulary Fragment.

On-line specifications written by student participants are being used to test the vocabulary.

4.1. Identification of the Concepts to be used by PHRAN

Digital systems contain one or more processes (independently executing environments) which compete and/or communicate. A process can be started asynchronously (whenever specified conditions become true); execute indefinitely; start, suspend and terminate other processes asynchronously; exclude other processes from executing; communicate with other processes; and be asynchronously terminated or suspended itself when some specified conditions become true. Clock rates at which processes run may vary from process to process. Processes communicate via shared data, synchronize at critical points, or compete for shared resources.

A taxonomy of high-level system behavior has been produced, and is shown in Figure 4-1.

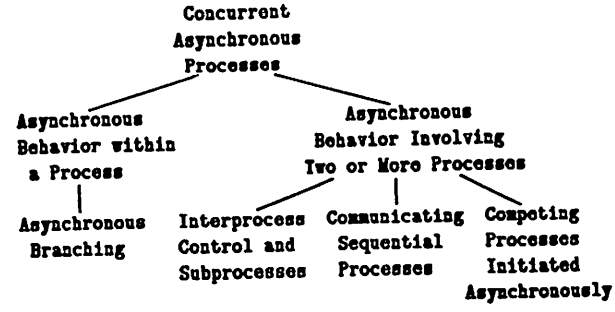


Figure 4-1: A taxonomy of High-Level System Behavior

An example of **asynchronous branching** is RESET. Asynchronous branching occurs when the current sequence of events within a process is altered or escaped from asynchronously in response to a change in a value of a logical predicate at some instant in time.

The RESET example has certain properties which require the DDS to contain special semantics. RESET is modelled in the control and timing subspace by a special predicate, attached to a time range, and is evaluated over the range as a continuous function of time. Effectively, each point of the alpha-omega arc is an or fork to the point α_R as shown in Figure 4-2[1].

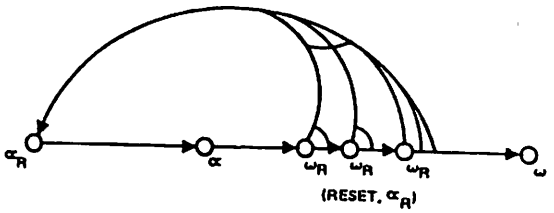


Figure 4-2: Representation of Asynchronous Branching

5. Conclusions

This research is an ongoing project. However, we can draw some conclusions at this stage. The DDS is useful for system specification, the use of PIIRAN for processing specification text is feasible, synthesis packages can use the DDS as both an input and output medium, semantic-net models can be used to capture design knowledge, and that a planning approach holds promise in the design of digital systems.

[1]The special points α and ω are used in the representation of loops. The point α is the starting point of a loop, and the point ω is the end of the loop.

References

[1] H. Afsarmanesh, D. Knapp, D. McLeod, and A. Parker.
An Extensible Object-Oriented Approach to Databasees for CAD/VLSI.
Submitted to the 11th International Conference on Very Large Data Bases.

[2] Breuer, M. and Zhu, X.
A Knowledge Based System for Selecting a Test Methodology for a PLA.
In *Proceedings of the 22nd Design Automation Conference.* ACM and IEEE, 1985.

[3] Breuer, M. and Chowdhury, S.
The Construction of Minimal Area Power and Ground Nets for VLSI Circuits.
In *Proceedings of the 22nd Design Automation Conference.* ACM and IEEE, 1985.

[4] H. Brown and M. Stefik.
Palladio: An Expert Assistant for Integrated Circuit Design. 1982.
Memo KB-VLSI-82-17 (working paper), Xerox PARC.

[5] Knapp, D. and Parker, A.
A Data Structure for VLSI Synthesis and Verification.
Technical Report DISC 83-6a, Digital Integrated Systems Center, Dept. of EE-Systems, University of Southern California, October, 1983.

[6] Fadi Kurdahi and Alice Parker.
Wiring Space Estimation of Standard Cell Designs.
Technical Report DISC/84-5, University of Southern California, Department of EE-Systems, November, 1984.

[7] Landman, B. and Russo, R.
On a Pin or Block Relationship for Partitions of Logical Graphs.
IEEE Transactions on Computers C-20:1469-1479, 1971.

[8] Park, N. and Parker, A.
Synthesis of Optimal Clocking Schemes for Digital Systems.
Technical Report DISC/84-1, Dept. of EE-Systems, University of Southern California, May, 1984.

[9] Parker, A., Kurdahi, F. and Mlinar, M.
A General Methodology for Synthesis and Verification of Register Transfer designs.
In *Proceedings of the 21st Design Automation Conference.* ACM SIGDA, IEEE Computer Society, June, 1984.

[10] Parker, A., Park, N. and Knapp, D.
Simulation Effectiveness and Design Verification.
Technical Report DISC/84-2, Department of EE-Systems, University of Southern California, October, 1984.

[11] Wilensky, R., Arens, Y. and Chin, D.
Talking to UNIX in English: An Overview of UC.
CACM 27(6), June, 1984.