A Model of Design Representation and Synthesis[1]

Robert A. Walker and *Donald E. Thomas*

Electrical and Computer Engineering Dept.
Carnegie-Mellon University

Abstract

To represent the increasingly complex designs being produced today, we have developed a unified model of design representation that uses three hierarchical, non-isomorphic domains of description that can be coordinated to represent the entire design. Each of these domains contains multiple levels of abstraction; both the domains and the levels are described in detail in this paper. We then show how this model of design representation can be used as a model of design synthesis. It is hoped that this work will lead to a better understanding of design representation and its relationship to the synthesis process.

Keywords: *Design Representation, Computer Hardware Description Languages, Design Synthesis.*

1. Introduction

The traditional approach to computer design representation, or design description and documentation, has been to describe a design using a single Computer Hardware Description Language (CHDL). Although the exact usage varies, most of these languages typically describe both the behavior and the structure of the design, possibly including physical or geometrical information as well. We feel that indiscriminately combining this information into one language does not adequately model today's increasingly complex designs, and have developed a unified model of design representation that separates the behavioral, structural, and physical information into three domains. These three domains can then be coordinated to represent the entire design. Furthermore, this model of partitioning domain information has been used to redefine the traditional (and sometimes ill-defined) levels of abstraction, bringing them more into line both with this model and current design practices.

Since synthesis tools are becoming increasingly important, we have also developed our *model of design representation* to serve as a *model of design synthesis*, illustrating the various synthesis tasks that might be performed. As these synthesis tasks proceed, they naturally create both increasingly detailed representations as well as additional inter-domain links as they transform a behavioral description to a structural description to a physical description. In this paper, we give a definition of design synthesis using this model, and show some examples of synthesis tasks.

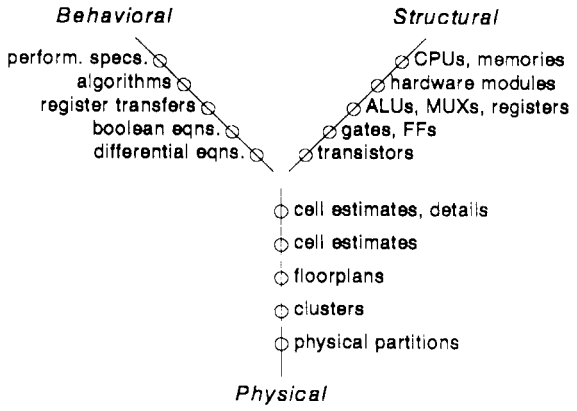

Figure 1.
Domains of Description -
An Axial View

2. An Overview of the Model

Our model of design representation can be described using three axes, each representing one of three *domains of description*, much the same as Gajski and Kuhn's *Y-chart* [1]. Along each of these axes, multiple *levels of abstraction*, or *levels of detail*, are represented. This view of our model is shown in Figures 1 and 2. The first figure shows three domains of description, named the Behavioral, Structural, and Physical Domains. These domains are represented by three axes approaching a common vertex, with the level of abstraction decreasing as one moves toward this vertex. The next figure depicts the same view, adding a standard set of levels of abstraction (shown as *rings*) that pass through all three domains.

[1] This research has been supported in part by the National Science Foundation under grant ECS-8207709 and the Semiconductor Research Corporation under grant 82-11-007.

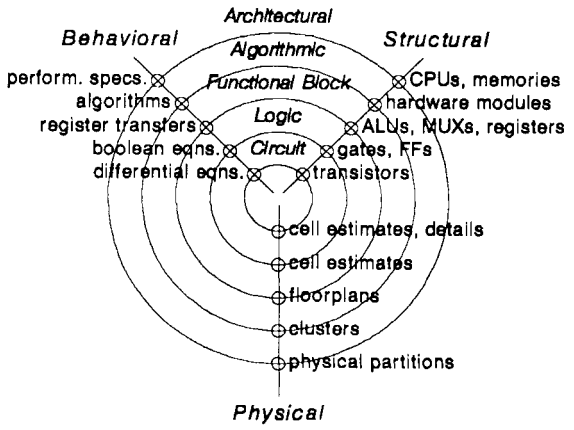

Figure 2.
Levels of Abstraction versus
Domains of Description -
An Axial View

	Behavioral Domain	*Structural Domain*	*Physical Domain*
Architectural Level	performance specs.	CPUs memories switches controllers busses	physical partitions
Algorithmic Level	algorithms (manipulation of data structures)	hardware modules data structures	clusters
Functional Block Level	operations register transfers state sequencing	ALUs MUXs registers microsequencer microstore	floorplans
Logic Level	boolean equations	gates flip-flops latches	cell estimates
Circuit Level	differential equations	transistors capacitors resistors	cell estimates cell details

Table 1.
Levels of Abstraction versus
Domains of Description -
A Tabular View

Table 1 shows an alternate, more familiar, view of this same model of design representation. It represents the model as a table with the *levels of abstraction* represented by rows, and the three *domains of description* by columns. As with the previous model, a design includes components from all three domains.

In this model of design representation, each domain is hierarchically decomposed into different levels of abstraction, and each decomposition is represented as a directed acyclic graph. Although these decompositions may also be represented as *trees*, this is not a necessary condition. Furthermore, the decomposition of one domain is not necessarily isomorphic to the decompositions of other domains. This is discussed in more detail later in this paper.

Other researchers have similar models of design representation; Gajski and Kuhn's *Y-chart* [1] and Knapp and Parker's *Design Data Structure* (DDS) [3] are two examples. We have combined what we feel are the best features from each of these two schemes, and added some of our own work to produce the model described in this paper. All three models are compared in a later section of this paper.

3. Domains of Description

In our model of design representation, there are three *domains of description* called the Behavioral, Structural, and Physical Domains. Each of these domains represents its particular portion of the total design description and must be coordinated with the other domains through inter-domain links. Although other divisions of domain information have been advocated (e.g., by Gajski and by Knapp), we feel that this division is the most natural. As with the modules advocated by Parnas [5], each domain is "characterized by its knowledge of a design decision which it hides from all others." Thus, the Behavioral Domain describes the basic functionality of the design, the Structural Domain the abstract implementation of the design, and the Physical Domain the physical implementation of the design. Other design decisions, such as those involving timing or topology, can be subsumed by the decisions above and are not represented by separate domains in our model.

3.1. Behavioral Domain

The Behavioral Domain describes the behavior, or *functionality*, of the design, and contains obvious static and dynamic components. The static component describes the operations, while the dynamic portion describes their sequencing and timing. Thus, differences in algebraic functionality, pipelining, and timing are all changes in behavior. For example, at the Functional Block Level, the Behavioral Domain describes the design in terms of register transfers, possibly including timing information. If the manipulation of the registers is changed, the static Behavioral Domain information is changed as well. Similarly, changing or introducing pipelining or timing information changes the dynamic behavior, and possibly the static behavior as well.

3.2. Structural Domain

The Structural Domain describes the logical structure, or *abstract implementation* of the design, usually as the structural inter-connection of a set of abstract blocks. *It is the mid-point between the Behavioral and Physical Domains.* For example, at the Functional Block Level, the Structural Domain describes the

design in terms of the abstract ALUs, MUXs, and registers needed to implement the register transfers required by the Behavioral Domain. Since this domain also describes the structure of the control part of the design, changes to the abstract implementation of the controller are also changes to the Structural Domain information. This includes changing the form of the controller from a PLA-based to a microcoded controller as well as changing the contents of a control ROM.

3.3. Physical Domain

The Physical Domain describes the *physical implementation* of the design, or the realization of the Structural Domain's abstract structural components with real physical components. At the upper levels of abstraction, this domain usually contains constraints on the physical partitioning or floorplanning of the design. At the lower levels of abstraction, this domain contains a description of real physical components and their geometry (cell detailing). As an example, at the Functional Block Level, the Physical Domain describes the floorplanning needed to implement the required abstract ALUs, multiplexors and registers. At all levels, speed, power, and area constraints are often part of the Physical Domain.

3.4. Further Characterizing the Domains

Each of these three domains of description is hierarchically decomposed into different levels of abstraction, and these decompositions are represented as acyclic graphs. Figure 3 shows an example of part of a RAM that consists of an array of cells, each cell containing the same collection of transistors, represented as an acyclic graph. Although the generalized acyclic graph may be more efficient for representing regular structures, many designs can be represented by trees (a constrained type of acyclic graph) as shown in Figures 4 and 5.

Furthermore, the decompositions in each of the three domains may be nonisomorphic, meaning that there is not a one-to-one mapping between the decomposition structures. For example, Figures 4 and 5 depict the Behavioral, Structural, and Physical Domain components of part of a design. Figure 4 shows the synthesis of an abstract addition operation, in the Behavioral Domain, with two Structural Domain components - an adder, to perform the actual addition, and an input MUX, to supply the adder with the proper inputs. Since a single Behavioral Domain element corresponds to two Structural Domain elements, the two domains are not isomorphically decomposed. Proceeding further with the synthesis, these structural elements might be implemented directly in hardware, with a one-to-one mapping between the Structural and Physical Domains. However, it might also be possible that, *based on the implementation technology*, it makes more sense to implement both the adder and subtractor as a single ALU. This is shown in Figure 5, and again the two domains are non-isomorphically decomposed. Thus, to represent a design as a coordinated representation of these three domains, it will be necessary to maintain three different decomposition structures.

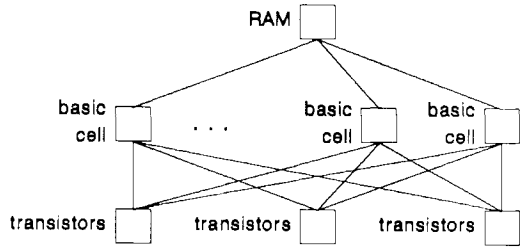

Figure 3.
Hierarchical Decomposition of a RAM

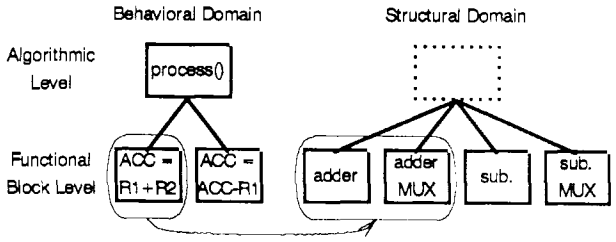

Figure 4.
Implementing an Addition Operation With an Adder and a Multiplexor

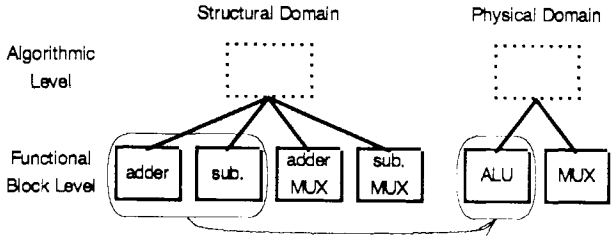

Figure 5.
Implementing Structural Operations In a Particular Technology

Each of the three domains may also have both a static and a dynamic component. The *static* component describes the (essentially) time-invariant portion of the description - the operations, structures, etc. Typical static components are addition and subtraction operations in the Behavioral Domain, and adders and subtractors in the Structural and Physical Domains. The *dynamic* component describes the ordering of the operations and the time between operations. Typical dynamic components are timing constraints in the Behavioral Domain, execution times for the abstract operators in the Structural Domain, and actual execution times in the Physical Domain.

Each of the three domains may also contain both a description component and a constraint component. The first component, the *description*, is a representation of the design as it exists at that point in the design process. This description may be provided by the user (e.g., as the input description for a synthesis system) or it may be built under program control as an intermediate or final representation of the design and linked to the user-supplied information. The second component of each domain, the *set of constraints*, restrains the design process. Timing constraints, area constraints, and power consumption constraints are typical examples. Although constraints are perhaps most often applied to the Physical Domain, they are not limited to that domain.

4. Levels of Abstraction

In our model of design representation, there are many *levels of abstraction*, each containing a Behavioral, Structural, and Physical Domain component. This section defines the uppermost levels in that model. Unfortunately, many of the names of these levels already have connotations that associate them with a single component of a particular level rather than the entire level. For example, the terms *Logic Level* and *Circuit Level* might suggest to many people only the structural component of those levels, and the term *Algorithmic Level* might suggest only the behavioral component of that level. Nevertheless, we have retained the more accepted terminology in our model, believing that inventing new names at this point would only further confuse the issue.

4.1. Architectural Level

The Behavioral Domain at this level describes the behavior of a system as a set of performance specifications, or gross operational characteristics. It is concerned with pieces of hardware that manipulate data and store results, without being concerned with the algorithms describing the details of how this occurs. In many instances, these performance specifications are supplied as a combination of an informal behavioral description and Physical Domain constraints.

The Structural Domain components that correspond to these performance specifications are processors, memory units, switches, buses, and device controllers. These elements are usually thought of as operating concurrently and communicating via ports or buses. In most formal descriptions at this level, the structural information makes up the bulk of the description, perhaps because of a lack of formal methods for describing the behavior.

In the Physical Domain, high-level physical partitioning of the design may be described. Depending on the implementation technology and the complexity of the design, this might correspond to chip or cabinet level partitioning. Besides the description, the Physical Domain might also contain constraints (e.g., power consumption, area, etc.).

4.2. Algorithmic Level

This level, often called the Behavioral Level, describes the design at a level syntactically similar to programming languages. Complex algorithmic expressions, data structures, procedures, processes,[2] and scoped variables are all used to describe the behavior of the design. Algorithms describe the necessary manipulation of data structures, and procedures, processes, and scoped variables provide the necessary code management. Examples of Behavioral Domain components at this level are instruction decoding, effective address calculation, and instruction execution.

In the Structural Domain, hardware modules (e.g., a separate data path and controller) represent the processes of the Behavioral Domain. These hardware modules are thought of as operating concurrently with each other, although their internal operation may include both sequential behavioral and statement-level parallelism.

In the Physical Domain, clustering of operators into physical subsystems might be described. Clustering, or partitioning, refers to grouping functionally similar operations together as determined by some measure of *proximity*. Operators with a high proximity tend to have a high potential for sharing hardware, and can be grouped together in the same physical subsystem. Besides this description component, constraints on power consumption, area, etc. may also be described.

Timing information may also become a concern at this level, as high-level behavioral timing constraints. For example, if the hardware being designed must interface to other hardware, it will be necessary to describe constraints on its execution time and on its interface timing. These constraints may either be described as part of the behavioral descriptions, or they may be described in a separate constraint language.

4.3. Functional Block Level

Often called the Register Transfer Level, this level describes the design at a level much closer to the underlying hardware than the previous levels. The Behavioral Domain contains either arithmetic or logical operations and transfers between registers (hence the name Register Transfer), sequencing between states, or a combination of both. A wide range of methods are used to describe this behavior, ranging from functional or dataflow languages through procedural languages to non-procedural, state-machine oriented languages.

[2] In hardware terms, each *process* is implemented by a separate data path and control unit.

In an abstract fashion, the behavior at this level is implemented in the Structural Domain using ALUs, adders, comparators, MUXs and registers. All these elements execute in parallel, and are usually regulated by a single control unit.

Depending on the target technology, it may be possible to implement this structural description directly in hardware. If this is not the case, or more detail is desired, the Physical Domain at this level might be concerned with floorplanning the layout of the design by specifying the geometrical arrangement of the Structural Domain components. Finally, as with the upper levels of abstraction, power and area constraints may also be part of this domain.

Since this level is much closer to the underlying hardware than the previous levels, timing plays a much bigger role than before. At this level, timing information consists of *major cycles*, or *machine cycles*, each corresponding to a major control state, and further subdivided into a fixed number of *minor cycles*, or *data flow cycles*. Timing information at this level may also contain references to discrete events, particularly in asynchronous designs.

Perhaps for the first time, the control part of the design is an explicit entity at this level, rather than being implicit in the behavioral description as before. Like the data portion, the control portion may have behavioral, structural, and physical components. At this level, the Behavioral Domain component describes the manipulation and sequencing of control signals, which in turn effects actions in the data part. The Structural Domain component, representing the abstract implementation, describes abstract structures such as PLAs and microcoded control ROMs (including the information stored in the ROM). The Physical Domain component is similar to that of the data path, requiring either implementation in hardware, or a part in the floorplanning process.

4.4. Logic Level

This level, often called the Gate Level, is a slightly more detailed version of the Functional Block Level. In the Behavioral Domain, it describes the design as a switching circuit, expressing the behavior in terms of boolean equations and finite automata. In the Structural Domain, this behavior is implemented by gates, flip-flops, and registers. All these elements are usually thought of as operating in parallel.

Like that of the Functional Block Level, the Physical Domain at this level may be directly realizable in hardware. For instance, in TTL-based design the Functional Block Level components without hardware counterparts might be hierarchically decomposed into gates, flip-flops, etc., which can be realized in hardware. Alternately, this level might be virtually ignored, with the designer proceeding directly from the Functional Block Level to the Circuit Level. For example, the implementation of the OM2 chip described in Mead and Conway's *Introduction to VLSI Systems* [4] proceeds directly from a Functional Block Level floorplan (in terms of registers, a shifter, and an ALU) to a Circuit Level implementation (in terms of transistors and inverters), ignoring the Logic Level. Finally, as with the upper levels, power and area constraints may also be part of this domain.

As the implementation of the design becomes increasingly detailed, timing descriptions and constraints may become more detailed as well, possibly in all three domains. At this level, timing may include references to arbitrary transitions, edges, or levels, and it may also include detailed information on setup and hold times, propagation delays, etc.

4.5. Circuit Level

In the Behavioral Domain, this level describes the behavior of the design in terms of electrical potential and current using differential equations. In the Structural Domain, it describes the structure of the design in terms of such circuit components as transistors, diodes, resistors, and capacitors, according to the implementation technology.

In the Physical Domain, two activities are dominant in VLSI design. First, *cell estimation* produces a rough circuit design in terms of approximate geometries and their placement, perhaps using stick diagrams. Second, *cell detailing* specifies the exact geometry and placement of the components, completing the design.

5. Design Synthesis

This *model of design representation* also serves as a *model of design synthesis*, using the multiple levels of abstraction and three domains of description defined in previous sections. Within this model, *design synthesis* is defined to be one or more of the following:

- a few-to-many translation from the Behavioral Domain to the Structural Domain,
- a few-to-many translation from the Structural Domain to the Physical Domain, or
- a few-to-many translation from a higher level of abstraction in one domain to a lower level of abstraction in that same domain, or in another domain.

This translation process is few-to-many, meaning that there may be many structural implementations of a particular behavior, and many physical implementations of a particular structure, but for a given physical implementation there are few corresponding structures, and for a given structure there are few corresponding behaviors.

In more practical terms, *design synthesis* is the process of translating a high level (of abstraction) Behavioral Domain description to a low level Physical Domain description. This synthesis process includes translating and building inter-domain links from the

Behavioral to the Structural to the Physical Domain, as well as adding enough additional detail to produce a low level description from a high level one. Thus, the end goal of synthesis is to produce a Physical Domain description at a low enough level to be implemented in hardware.

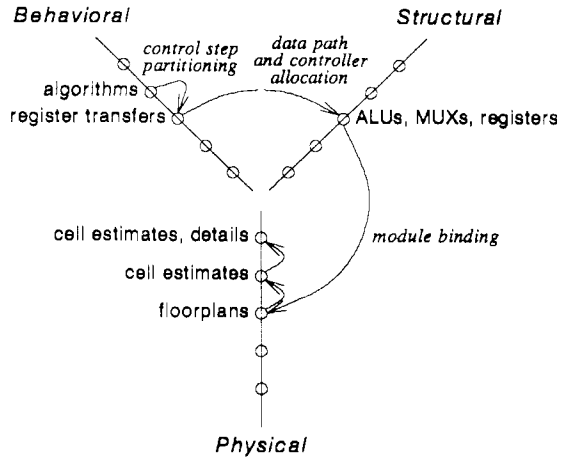

Figure 6.
Synthesis Tasks

Figure 6 shows some examples of synthesis tasks in the CMU-DA [6] design methodology. The first task, control step partitioning, is a translation from a high level of abstraction in the Behavioral Domain to a lower level of abstraction in that same domain. The second task, data path and controller allocation, is a translation from the Behavioral Domain to the Structural Domain at the same level of abstraction. The third task, module binding, is defined similarly but involves the Structural and Physical Domains. All three tasks are few-to-many translations in the sense that, for example, for a given set of register transfers (Behavioral Domain) there are many sets of ALUs, MUXs, registers, etc. (Structural Domain) that could be used to specify the abstract implementation of those register transfers. Choosing one of these many alternatives is exploring the design space.

This definition of synthesis also implies that, at a given point in the synthesis process, a design is represented by one or more components from each of the Behavioral, Structural, and Physical Domains, and may not fit neatly into any of the levels of abstraction defined earlier. For example, during the control step partitioning task of Figure 6, the design exists at multiple levels of description (both the Algorithmic and Functional Block Levels), but after the task is completed, the design exists at a single level, although it combines domains from two of the levels defined in Section 3 (i.e., it has a Functional Block Level behavioral component, but does not yet have a structural or physical component at that level). At either time, the design is still represented by one or more components from each of the three domains, although it does not fit neatly into any of the levels defined earlier.

6. Comparison to Other Work

Other researchers have ideas similar to ours. This section describes some of that work.

6.1. Gajski and Kuhn

Gajski and Kuhn [1] have developed a tripartite representation of design called a *Y-chart*, on which our axial view of domains of description (Figure 1) is based. Their *Functional Representation*, *Structural Representation*, and *Geometrical Representation* roughly correspond to our Behavioral Domain, Structural Domain, and Physical Domain, respectively, although our Physical Domain also includes non-geometrical information as well (e.g., actual area and power consumption).

A major addition to their work is our definition of a standard set of levels of abstraction, defined in terms of the domains of description. Other additions include our views of non-isomorphic, hierarchical decomposition, some of our views of design synthesis, static and dynamic components in each domain, and description and constraint components in each domain.

6.2. USC Expert Synthesis System

Currently under construction at the University of Southern California, the Expert Synthesis System (USC-ESS) [2] uses a data structure called the Design Data Structure (DDS) [3] to represent the design. This data structure was developed to provide a common, uniform representation for a set of DA tools (the ESS system), and was developed to provide a set of mutually orthogonal subspaces. They use the analogy of an architect's orthographic projection of the top, front, and side views of an object to derive these subspaces, or views, of the design such that changing a component of one subspace has a minimal effect on the other subspaces. For example, changing the physical partitioning of a design (e.g., moving components from one chip to another) has a minimal effect on the other subspaces.

The four subspaces used in the DDS are the data flow behavior subspace, the structural subspace, the physical subspace, and the timing and control subspace. Taken together, the *data flow behavior subspace* and the *timing and control subspace* correspond roughly to our Behavioral Domain. The data flow behavior subspace represents functional definitions and data dependencies between values as a data flow graph. The timing and control subspace, also called the sequencing subspace, represents the ordering of events, the conditional sequencing of events, and the time between events. The third subspace, the *structural subspace*, represents the electrical topology of the design, and is similar to our Structural Domain. Likewise, the *physical subspace* is similar to our Physical

Domain, encompassing both geometrical and purely physical information (e.g., power consumption).

In some ways, our work is closer to this work than that of Gajski and Kuhn, as both our representation and the DDS have hierarchical, non-isomorphic domains. However, there are also some major differences, the most basic being that our work was developed as an *model of design representation and synthesis*, whereas the work at USC was developed as a *design data structure*. Thus, we have a standard set of levels of abstraction that are consistent across all domains, and we have separated all three of the domains into a static and a dynamic component, and into a description and a constraint component.

7. Conclusion

We feel that a better understanding of design representation and its relationship to the synthesis process is needed, and we are trying to develop better models to fulfill that need. As designs become increasingly complex, it will no longer be possible to represent a design with one all-encompassing language. Rather, it will become necessary to divide the design description into large, separate domains of description that can be coordinated to represent the entire design. This paper has described such a model of design representation; it uses three coordinated, hierarchical, non-isomorphic domains of description, each of which contains multiple levels of abstraction. Furthermore, since synthesis tools are becoming more important, we have shown how this model can be used as a model of design synthesis, representing the task of design synthesis as a translation between domains and levels.

References

[1] D. Gajski and R. Kuhn,
Guest Editors' Introduction: New VLSI Tools,
Computer 16(12):11-14, December, 1983.

[2] D. Knapp, J. Granacki, and A. Parker,
An Expert Synthesis System,
Proc. of Int. Conf. on CAD, pages 164-165,
IEEE, Santa Clara, Calif., September, 1983.

[3] D. Knapp and A. Parker,
A Data Structure for VLSI Synthesis and Verification,
Report DISC/83-6a, Dept. of EE-Systems,
USC, March, 1984.

[4] C. Mead and L. Conway,
Introduction to VLSI Systems,
Addison-Wesley, 1980.

[5] D. Parnas,
On the Criteria To Be Used in Decomposing Systems into Modules,
CACM, 15(12):1053-1058, December, 1972.

[6] D. Thomas, C. Hitchcock, T. Kowalski,
J. Rajan, and R. Walker,
Automatic Data Path Synthesis,
Computer, 16(12):59-70, December, 1983.

The authors can be reached at:
Dept. of Elect. and Computer Engineering
Carnegie-Mellon University,
Pittsburgh, PA, 15213,
or via the ARPANET as:
Bob.Walker@cmu-ee-faraday.arpa and
Don.Thomas@cmu-ee-faraday.arpa.