# A Decision Support Environment For Behavioral Synthesis

Tedd Hadley and Daniel D. Gajski

Technical Report 91-17
August 26, 1991



Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717
(714) 856-8059

hadley@ics.uci.edu

**Abstract**

We present a specification of a general environment for behavioral synthesis centered around the user/designer as the primary motivator for decisions in design development. At each stage of the design process, the user can perform transformations on the design description through graphical user interfaces. Quality measures, physical estimates, and design hints are given to the user at each stage.

# Contents

# List of Figures

# 1 Introduction

This report describes the requirements and specifications for a decision support environment for behavioral synthesis. From this report, we hope to derive a general framework for a design environment fully supporting user decisions.

Two observations justify the need for a decision support environment for behavioral synthesis:

1. Complete automation of the design process through several levels of abstraction is not an immediately practical goal.

2. The human designer's insights into design strategy should be used to maximum effect in all phases of behavioral synthesis.

The first point is a conclusion derived more from the lack of observable uses of automated behavioral synthesis systems in commercial domains, rather than from the lack of existing systems in research and academia. Although it certainly can not be denied that progress has been considerable in this research area [Sh89] [DeRa86] [BrCa88], a practical solution to the problem of automating behavioral synthesis is still distant [CaWo91]. To develop a feasible approach to the problem, we have substituted the goal of a completely automated, "push-button" synthesis system with one which attempts to maximally utilize the human designer's methods and experience.

There are two primary goals in the development of our environment for decision support in behavioral synthesis:

- To allow user decisions and user control throughout every phase of the design process, and

- To provide rapid feedback of useable physical design characteristics and quality measures to every level of design abstraction.

Decisions made by the user must generate immediate feedback as to the quality and functionality of the resulting design. As the design process progresses to more detailed and less abstract design representations, the user's experience is utilized to make the kinds of decisions no automated tool can perceive or predict.

# 2 Previous Work

Research in behavioral synthesis has mostly been concentrated on specific tasks, such as scheduling, allocation, and binding. Initially, a high level description is compiled into a control/data flow graph. Scheduling slices the flow graph into state intervals, such that the operations described in each state will be executed in a single control step of the design implementation. A control step corresponds to one clock period. Allocation determines what hardware components will be required to implement the set of operators, variables, and connection paths in the design description. Binding assigns the components allocated to operators, variables, and connections, and also determines where selectors, such as multiplexors or buses, will be needed.

There are a large number of algorithms for these tasks which work quite well in theory [PaPM86] [THKR83] [PaGa87] [TsSi83] [DeNe89], but in practice, their relatively simple cost functions often do not take into account enough of the physical design layout characteristics to generate good results. Thus, they perform well on small benchmarks, but do poorly on real industrial designs.

Development of CAD frameworks has been mainly concentrated on databases and environments for the development of designs at the layout level.

We want to extend framework development to behavioral synthesis, at the same time providing good physical estimates in the form of back-annotation from layout levels to behavioral descriptions. Our user interfaces allow full exploitation of this information with quality measures displays, editors for different levels of design abstractions, visual links between abstractions, and design hints at each phase.

# 3 A Decision Support Environment

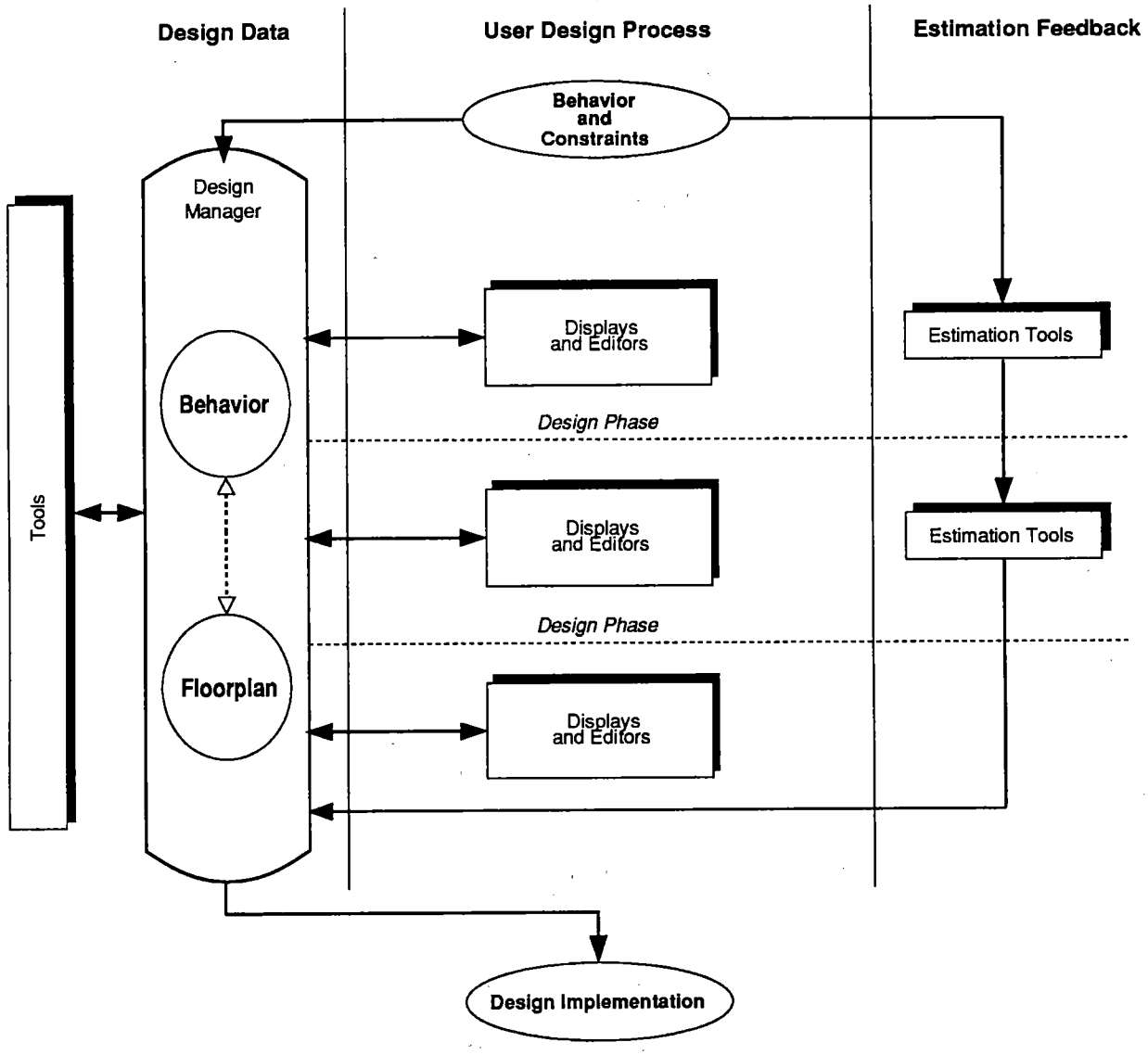



Figure 1: A General Decision Support Environment

A model of a general decision support environment is shown in Figure 1. The user dominates the design process at the center. The estimation-feedback mechanism, at the right, rapidly calculates measures of quality and stores them back with the design data. On the left, a system of data organization keeps track of the design representation at all levels of abstraction and maintains and

enforces links between behavior and structure. In each design phase, the user interacts with the environment through displays and editors. Simple tools, at the far left, may perform tasks under the supervision of the user, but need only communicate directly or indirectly with the central design representation.

## 3.1 The User Design Process

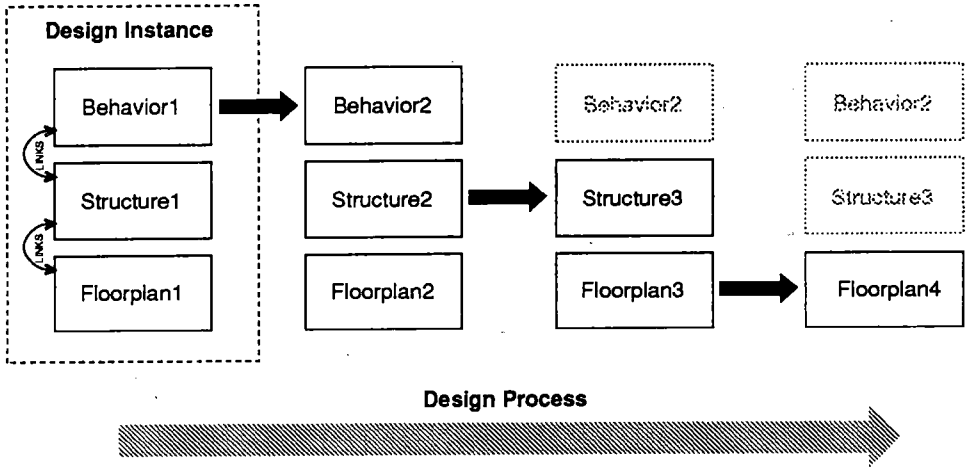



Figure 2: The User Design Process

In general, design development describes the path from the initial behavioral description to the final floorplan or layout. In our decision support environment this path is divided into distinct entities called design instances. Figure 2 shows how multiple design instances make up the design process. Each design instance contains a partial structure and floorplan representation of the specified behavior, although at early stages of the design process the implementation is not expected to be detailed or satisfactory. There are detailed links between the behavior, structure and floorplan of each design instance.

Initially, the user may iteratively modify the behavior and observe the changes on the estimated structure and floorplan. After further improvements are not observed, or behavioral optimizations become less obvious, the user may stabilize the behavioral description and begin to iteratively modify the structure and floorplan description to result in the final implementation.

## 3.2 The Design Database

The design database stores the various abstractions of the design representation in secondary storage. Any representation can be retrieved from the design database at any time by a tool or user interface.

The design database must also coordinate the various revisions of the design during the design process so that a complete history of the synthesis process is maintained. In addition, a single design is allowed to follow multiple development paths. Therefore, the design database must be able to keep these paths separate.

## 3.3 Component Database

The component database manages all available hardware components. Each component description must contain behavioral attributes (such as functionality) and physical characteristics (such as port names and locations, width and height or aspect ratio, delay, and area).

## 3.4 Design Manager

The design manager is the central controller for the environment. It spawns estimation tasks, tools, and displays on request, and handles the low level process communication required in an integrated environment.

The front-end to the design manager is the only part of the environment's framework that the user sees directly.

The remainder of this report discusses the user interfaces involved in the design process.

# 4 Design Data Representation

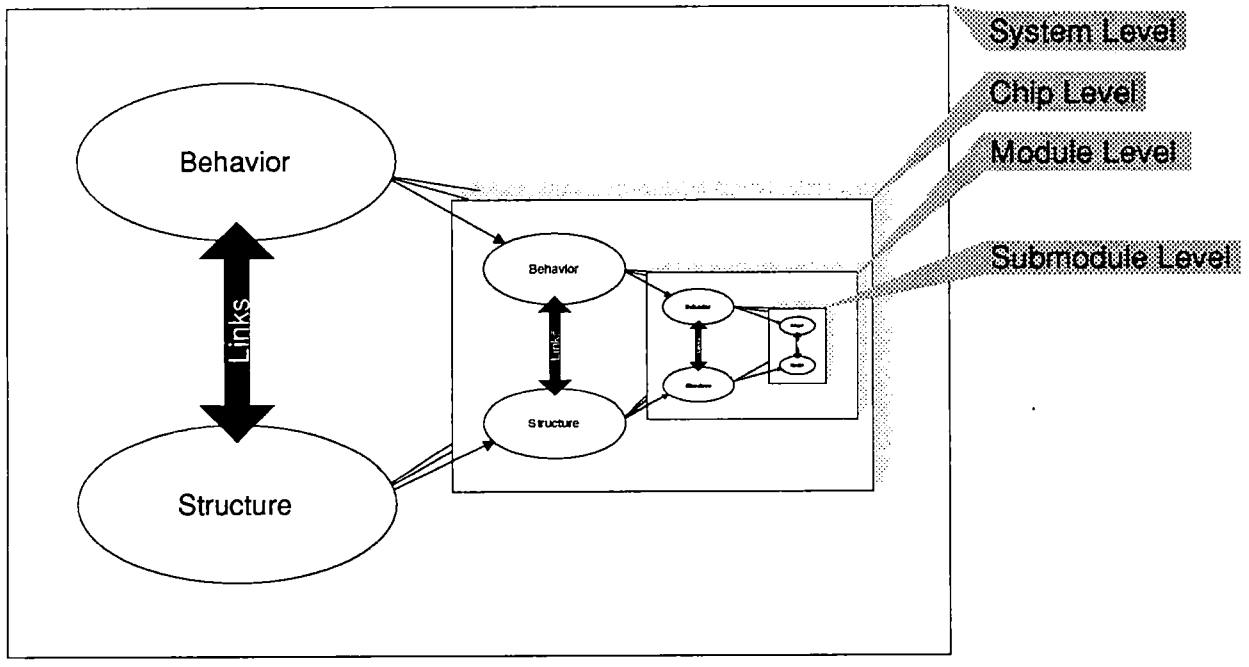



Figure 3: Hierarchical Design Representation

The design representation contains the behavior and structure of each design instance, and maintains the relationship between them in the form of links. Figure 3 describes the representation of a single design instance. The hierarchy of system descriptions is should be supported directly in the representation by the use of links to sub-descriptions.

Initially, a design instance is composed of one or more module descriptions. Each module description may describe behavior, structure, or both. Each description may in turn be hierarchical.

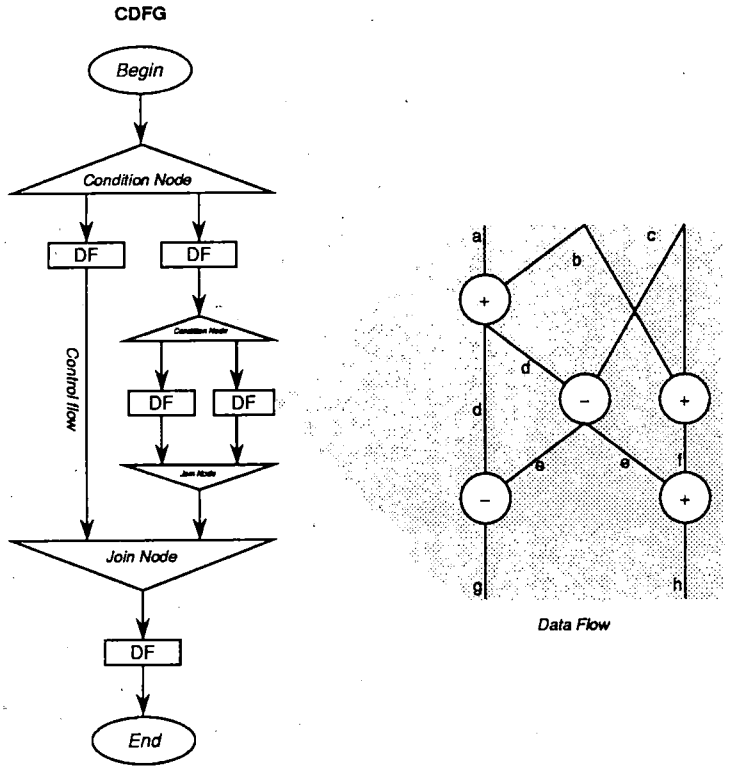



Figure 4: Control/Data Flow Graph

A design instance with completed structure allows partitioning of its sub-modules to meet design constraints. The links from behavior to structure facilitate user and tool decisions at the behavioral level during later phases of the design process.

## 4.1 Behavioral and Structural Representation

The basic behavioral and structural representation of the environment is the Control/Data Flow Graph (CDFG) [RuGa90] [RuGa91].

A CDFG describes the behavior of a design in a representation combining the flow of control with data flow activity. There are two kinds of control flow program behavior supported by the CDFG: *structured* and *unstructured.*

A structured control flow program requires that all branches and loops terminate at a single unambiguous "joining" point. Implicit in this description is the limitation that it is not possible to branch out of a loop, or to branch to a previous stage of the control flow graph. The primary goal of the structured style is to generate easily understood descriptions.

On the other hand, an unstructured control flow program allows completely unrestricted branching, similar to a state transition graph.

Though in general an unstructured representation is clearly a superset of a structured one, it may be necessary to preserve structure in the cases where translation from a structured control flow to a structured language is required. In addition, many algorithms for behavioral optimization have been developed which specifically rely on the structured nature of control flow to operate correctly.

Just as software programming languages encourage structured programming, but usually allow

the use of unrestricted branching via "goto" statements, so also does the CDFG representation support both programming models.

### 4.1.1 The Language Interface to the CDFG Representation

There are two intermediate formats acting as counterparts to the environment's internal design representation and oriented primarily towards user interaction. Both formats are based on the Behavioral Intermediate Format (BIF). The first, structured BIF, serves as a language for easy manipulation of structured control/data flow graphs. It preserves the structured nature of the control/data flow graph it represents. The second, unstructured BIF (or just BIF), provides a more powerful, though less structured, low-level interface to the design. It removes all restrictions on control flow, using a finite state machine model to represent the design.

The process of converting a CDFG to a structured BIF description involves folding the branches of control into a table of sequential states. Figure 5 shows the correlation between a control/data

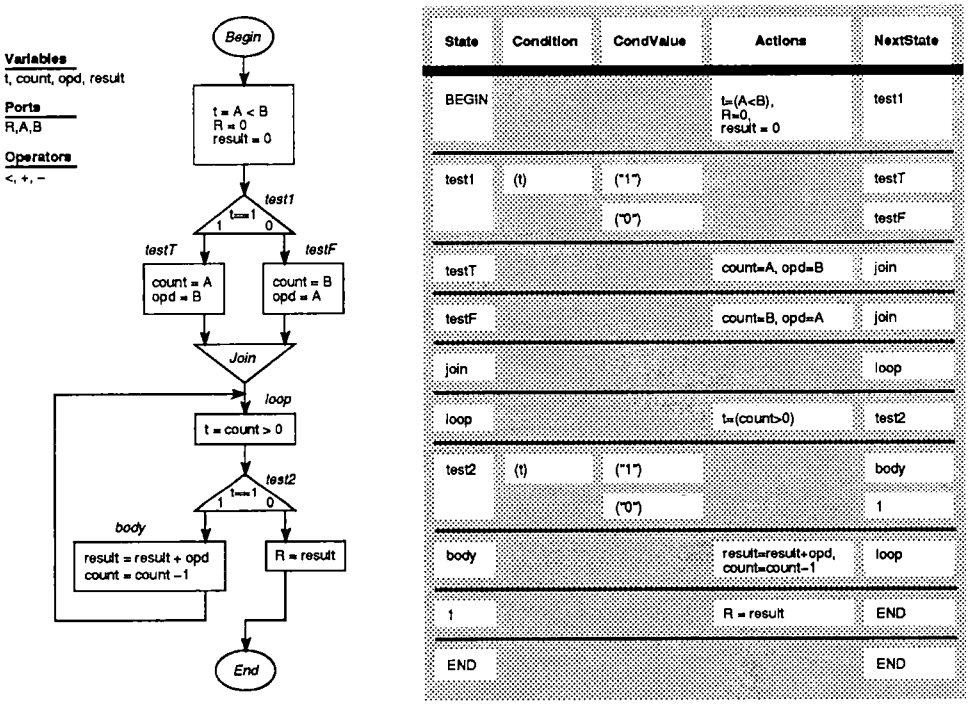



Figure 5: CDFG to Structured BIF

flow graph and a structured BIF description. Each node in the CDFG corresponds to a state in the intermediate form. The operations in a data flow statement block are described in the actions field of the corresponding state. Control flow conditions are represented by states that select various next states based on the values of the conditions. The states composing the exclusive branches of a condition node all transfer control to a single joining state, corresponding to the join node at the bottom of the control flow branches.

At the lower levels of abstraction a structured BIF description can be converted to unstructured BIF by collapsing operations states into condition states and removing joining states. The collapsed version of Figure 5 is shown in Figure 6.

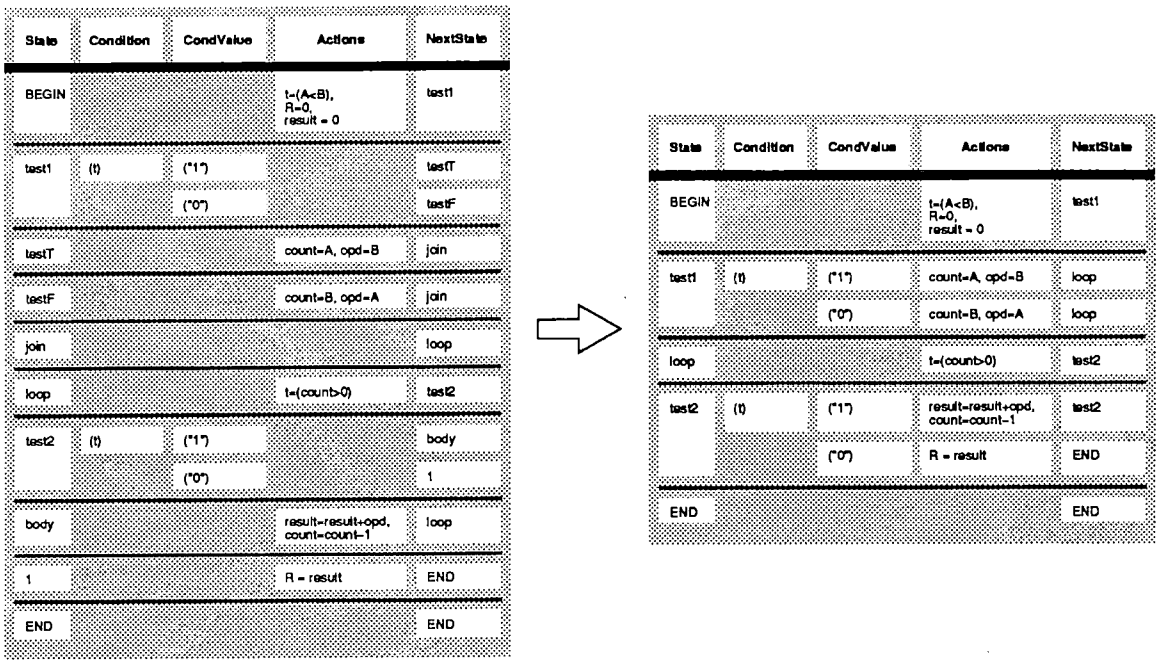



Figure 6: Structured to Unstructured BIF

BIF, either structured or unstructured, makes use of modified state tables annotated with a list of symbols, units, connections, and floorplan to describe a design at each level of abstraction in the synthesis process. Since a design spans several levels of abstraction, BIF has a slightly different format for each abstraction level in the design process.

The symbol list maintains information about behavioral variables, ports, and component bindings. The symbol table for the example in Figure 6 is shown in Figure 7. The unit list contains a detailed description of each component used in the design. The connection list stores the structural connectivity of the design. The floorplan list keeps track of the geometries of modules and wires, and the positions of ports.

The state tables and associated information lists are progressively updated as the synthesis process proceeds. At each level of abstraction the user can, either directly or through the use of tools, modify the state table and/or any of the information lists.

BIF differs syntactically from conventional software languages in that it is a strictly tabular format. Each row in the table represents a state, and the columns show the fields of each state. XBIF is a graphical editor which preserves this tabular form by displaying a BIF description on a workstation screen as a table of "form boxes". Each form box can be manipulated or modified by the user with the workstation keyboard and mouse. XBIF allows addition and deletion of states, quadruplets, and next state event pairs. Each field can be opened to allow standard textual editing commands.

BIF supports resource bindings by allowing behavioral variables and operators in the behavioral description to be explicitly bound to hardware components. Figure 8 shows an example of operator binding. The operation described at the top has the unit instance *ALU1* associated with the $+$ operation. The unit list defines the instance as belonging to the component *ALU8* which in turn is

```
SYMBOL TABLE {

     TYPE
          Tinput     = {7..0};
          Toutput    = {7..0};
          Tbit       = {1};

     PORT
          A,B        = INPUT of Tinput;
          R          = OUTPUT of Toutput;

     VAR
          result     : Toutput;
          count,opd  : Tinput;
          t          : Tbit;
}
```

Figure 7: BIF Symbol List

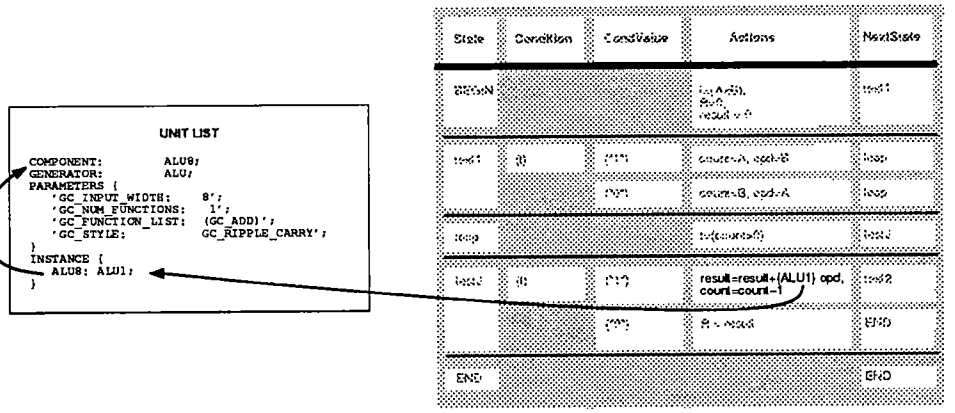



Figure 8: Operator Binding in BIF

a sub-type of the generator *ALU*. The parameters of the component specify that it is a 8 bit ALU, performing a ripple-carry ADD function.

At the structural level, BIF describes the units that are active in each state, the functions they perform, and the inputs to each unit. Figure 9 shows a behavioral action from the table in Figure 8

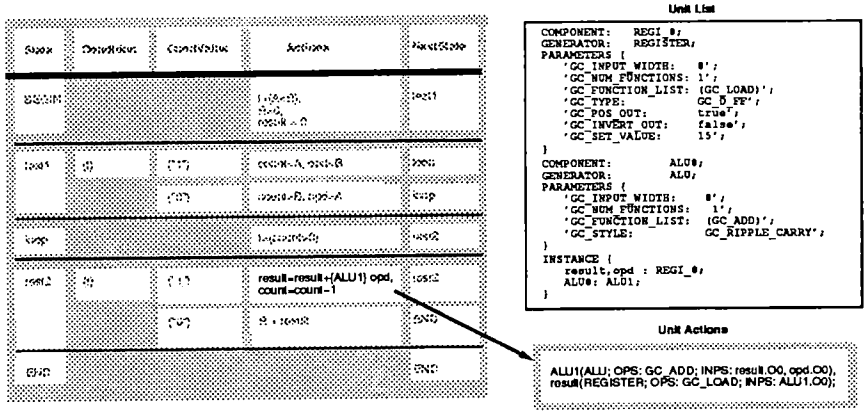



Figure 9: Operations Related To Units in BIF

on the left. At the right is the corresponding unit-based BIF action entry. Above the entry is the unit list describing three of the instances used: *result*, *opd*, and *ALU1*. A unit action contains the instance name, the operation performed by that instance, and the inputs to that instance.

From such a description, a control table can also be derived listing the values of the component control pins for each state. With minor modifications, the table can then be used as input to control synthesis.

# 5 User Interface

The user interfaces to the environment support display and modification of the design at different levels of abstraction, and visually describe relationships between different abstractions of the same design. At each stage, the design manager extracts and maintains quality measures. These measures are used to provide motivation and direction for user decisions in the design process.

## 5.1 Input Constraints

In most cases, the designer has some idea of the desired area and performance for the goal design. Information about the kinds of tested units available for the layout technology are known. Specific timing and area constraints need to be given for parts of the behavioral description. Even connectivity details may be known before the design process begins. These constraints need to be easily specified to the environment, communicated to tools, and enforced throughout all internal and external decisions made in any automated processes supported by this environment.

Units and connectivity constraints are usually described as partial bindings and partial structure. BIF supports partial binding by allowing a mixture of unit binding with the behavioral operations in a behavioral description, as described in Section 4.1.1. Partial structure must, of necessity, be

accompanied with a degree of unit binding for a point of reference to the behavior, so BIF allows the bound units to be referenced in a connectivity list associated with the behavioral description. The bindings and connectivity list are employed as both a starting point and a fixed constraint during unit allocation and connection binding.

In BIF, duration constraints can be placed on operations or states. An operation can be constrained to complete after a specified amount of time or after a certain amount of time after the event that caused the transition to the state occurs. States can have a time–out or duration event which causes a transition to a specified next state after a certain amount of time has expired. Figure 10 gives several examples of these timing constructs.

| State | Condition | CondValue | Actions | NextState | Event |
|---|---|---|---|---|---|
| | | | A \| (DELAY 10ns AFTER clock RISING) = B + C; | | (AFTER 10ns) |
| | | | A \| (DELAY 10ns) = B + C; | | (AFTER MAX 10ns) |
| | | | A \| (DELAY MAX 10ns) = B + C; | | (TIMEOUT 10ns) |
| | | | A \| (DELAY MIN 10ns AFTER clock RISING) = B + C | | |

Figure 10: BIF Timing Examples

Since a design is initially partitioned into hierarchical behavior, each entity in the description may be assumed to be a separate piece of hardware. Therefore, it is possible for any of the various parts of the design to have area constraints. The constraints themselves may be fixed ($area == 1000\ microns^2$), dimension sensitive ($width == 250\ microns$, $height\ variable$), or range specific ($750 < area < 1000$).

Area constraints are specified in the floorplan display by a bounding box. The relative sizes of other components and routing with respect to the bounding box will then be displayed in the floorplan display. Units or routing area overlapping the bounding box visually communicate trouble spots to the user.

## 5.2 Interactive Synthesis Tasks

This section describes several interactive synthesis tasks supported by the environment. The tasks are design description, unit selection, general unit assignment, connection allocation and binding, module and port placement, scheduling, and detailed unit assignment. Each task is performed by the user with the aid of interactive display tools, as shown in Figure 11. In addition to providing the editing mechanisms necessary to perform each individual task, the display tools coordinate and communicate information about the quality of the design represented.

### 5.2.1 Unit Selection

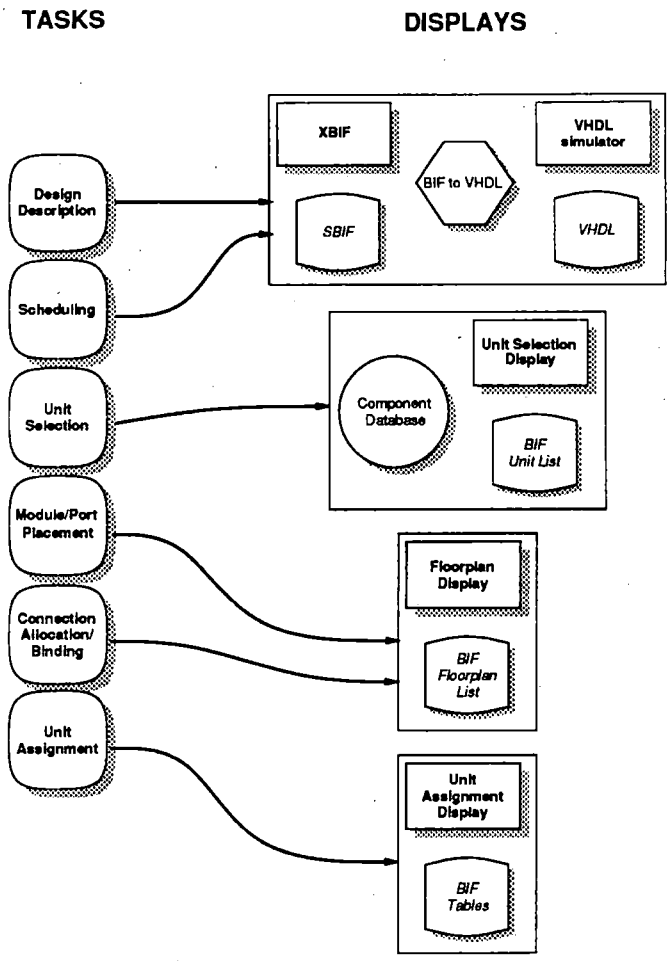



Figure 11: User Tasks and Displays

Working in conjunction with the component database is a tool to select components for the design implementation (shown in Figure 12). The available components in the component library can be browsed by increasingly specific categories, such as class (combinatorial or clocked), the general classification of the component (ALU, comparator, decoder), the available bit widths, the functions performed by the unit, and the style (ripple carry, for example). Physical information about each component is displayed once the previous categories have been specified. This information includes width and height of the component, and nominal, maximum, and minimum input–to–output pin delays. Physical characteristics are estimated based on specific technology library components. The user may select desired components, which are then stored in the set of allocated components for the current design. The current set of allocated components is always maintained in the BIF unit list (described in Section 4.1.1).

After unit allocation, the widths, heights, and port locations of the selected components provide enough information to allow the user to perform initial floorplan placement. Described in the next section, the floorplan display permits this task to be done graphically.

### 5.2.2 Floorplan Placement

The placement task determines the physical positions of the various modules composing the design. The different kinds of modules may include memories, data paths, glue logic, or macrocells.

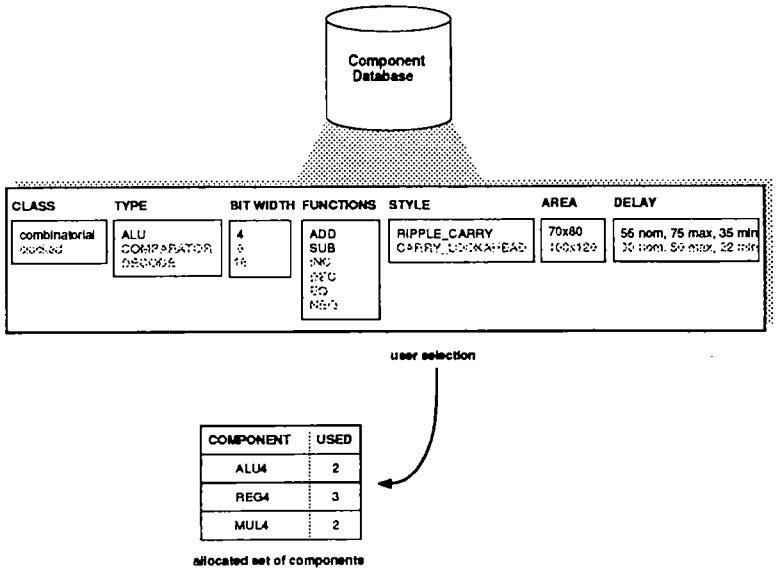



Figure 12: Unit Selection

The floorplan display (Figure 13) shows the sizes and positions of modules and components in the design. Connections are shown "point to point" between module ports. Within a data path/control module, the storage, functional, and interconnect units, are shown along with a block representing the control. The bit–sliced components in a data path are arranged vertically with internal connections displayed to the right and left of the component stack.

In general, the floorplan display allows adding, deleting, and repositioning of modules. Ports can be moved along the perimeter of a module.

The specific commands supported by the interface are

- Module deletion: a module and its connections are removed from the display. The netlist maintained in the design's connection list is not changed until a new module is added to perform the old module's function.
- Module addition: a new module is placed in the design's floorplan.
- Module rotation: a module is rotated left or right by ninety degrees.
- Module reposition: a module is relocated to a new position not overlapping any other module. Its connections are not changed.
- Port repositioning along an edge of a module: a port is moved in either direction along an edge of a module. The port's connections do not change.
- Port repositioning to a different side of the module: a port is moved to one of the three other sides of its module. Its connections are not changed.

At any time several quality measures can be supplied to the user on demand. These measures are total area, longest wire, total wire length, channel density with respect to wire pitch, wasted area with respect to total wire area, an estimate of delay for any wire, and the longest delay path.

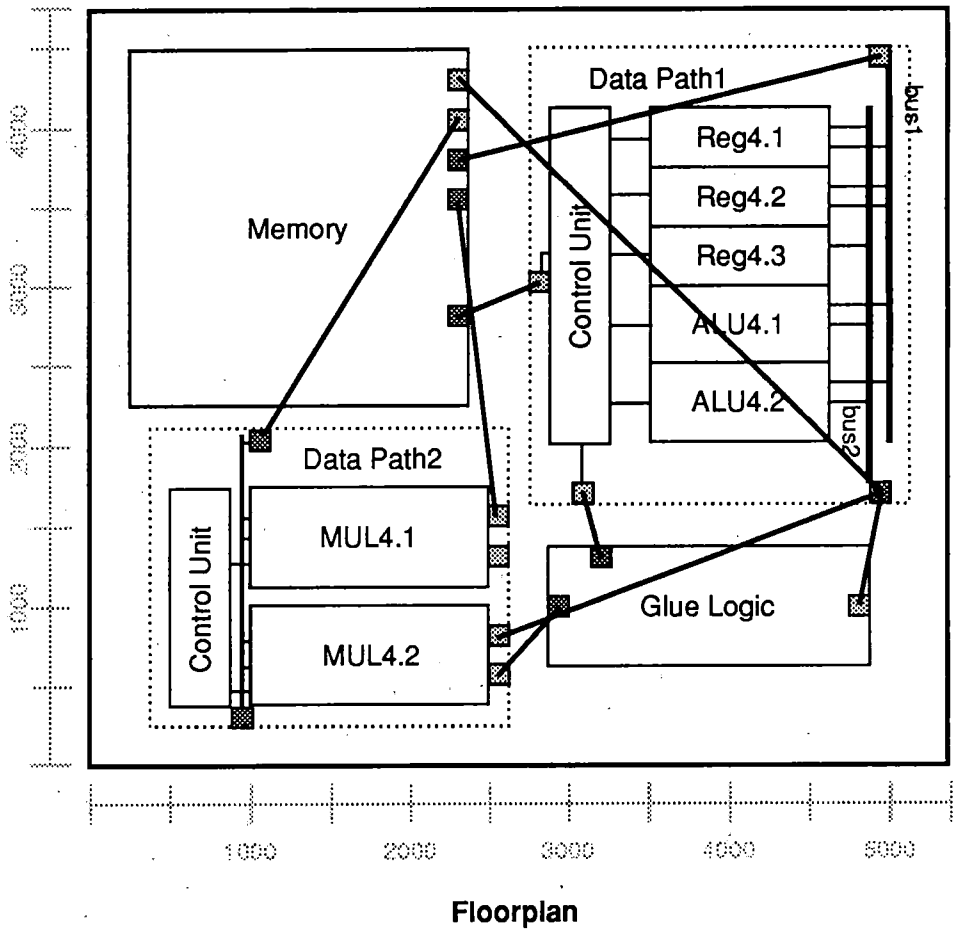



Figure 13: Floorplan Display

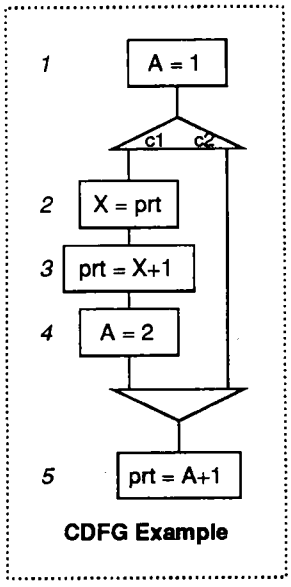
There are also three sub–tasks built in to the floorplan display that perform algorithms or heuristics on sections of the design in order to give the designer hints as to possible improvements. The first finds the geometric center for all ports on a single specified net and attempts to position them as close to that center as possible. The new port positions will result in the minimum wiring area for that net. The second finds the best port positions for two selected modules by locating the nets that the two modules have in common and determining locations for the module ports on these nets that result in the least routing between the modules. The third automatically tries each of the four rotations on a given module and returns the rotation that gives the minimum routing area around that module.

The designer may use the floorplan display to initially position components selected from the unit selection display. During unit allocation, unit assignment, and connection binding, the designer may wish to reposition modules and relocate ports based on the quality measures provided.

Floorplan information is maintained in the BIF floorplan list (described in Section 4.1.1).

### 5.2.3 Unit Assignment

The unit–assignment display is a user interface to perform and analyze unit assignment. The designer uses this display to observe the utilization of initial unit bindings specified in the behavioral description (described in Section 3), to assign variables or operators to newly allocated units, and to modify current bindings by reassigning variables or operators to different units.




| | 1 | 2 | 3 | 4 | 5 | Inputs | Utilized |
|---|---|---|---|---|---|---|---|
| R1 | c1 +A c2 | | | +A | A– | 2 | 75% |
| | | A | A | A | | | |
| R2 | c1 c2 | +X | X– | | | 1 | 25% |
| INC1 | c1 c2 | | +1 | | | 1 | 12% |
| INC2 | c1 c2 | | | | +1 | 1 | 12% |

*Track Density:* 4

Figure 14: Unit Assignment Display

Figure 14 shows how the information is organized for interactive resource utilization analysis. The CDFG for this example is shown at the left. The resource utilization display shows a single resource per row of the table. Storage resources are listed first, followed by functional units. The columns of the table correspond to the states of the design. Each resource entry is divided horizontally per state into blocks according to the number of exclusive control flow branches at that

point. The conditional expression is shown in an empty block preceeding the branch. If a variable is stored in a register, or an operation performed by a functional unit in a given branch and state, it is named in that block. For variables in storage units, the prefix + indicates a write to that variable, while the suffix - indicates a read.

Variables may be assigned or moved vertically between resource entries if the target storage resource is free in that state. In the example shown, it is possible to move the assigned variable *X*, stored by register *R2* in states 2 and 3, into the empty slots owned by register *R1*. The result makes R2 unneeded in this design. Operators may also be assigned or moved between resource entries if the target functional resource is free and able to perform the desired function. For example, the increment operation performed by *INC2* in state 5 can be moved up into the empty slot owned by *INC1*, making only one incrementer necessary.

The final two columns in the table list the number of inputs to each resource, and the resource utilization amount as a percentage of the total number of exclusive states. The input estimate corresponds closely to the number of multiplexors or buses required to implement the design. For a given storage unit, its value is determined by summing the number of states in which one of its variables becomes active (is written). For functional units, the number of states where the operation is performed determines the value.

At the bottom is given a track density estimate, calculated from the floorplan information by stacking the current set of components and counting each input to output as a single track.

The commands supported by the resource–display interface are

- Resource addition: a resource entry is added to the display after it has been selected with the unit–selection interface.
- Resource deletion: a resource that has no assignments in any state is deleted. The corresponding resource in the design's unit list is also deleted.
- Resource assignment: a variable or operator tag is assigned to a resource in the column corresponding to the state in which it is to occur.
- Resource reassignment: a behavioral variable or operator tag is moved vertically between resource entries if the target resource is free and able to perform the necessary function.

The unit–assignment display can also "compress" the amount of information displayed to a ranges of states, a single state, or a single "wildcard" entry. This may be done for two reasons. First, early in the design process, the user may wish to specify that certain variables be permanently assigned to unique storage elements, and that certain operators be bound to specific functional units. In order to do this, the user indicates to the unit–assignment display that all states should be compressed into a single entry. A variable or operator name may then be entered in the row corresponding to the desired resource, and the resulting binding will take effect across all states, where possible. In Figure 15, the display shown at the bottom would be a reasonable initial binding for the resources shown. Later, after more detailed binding, the display at the top would show the exact bindings of each resource in each state. A second reason for state compression is to allow the user to focus on a particular state or range of states. Since the amount of information displayed for a design with either a large number of resources, a large number of states, or both may be quite voluminous, it is important to let the user constrain the display output to desired problem areas or "hotspots".

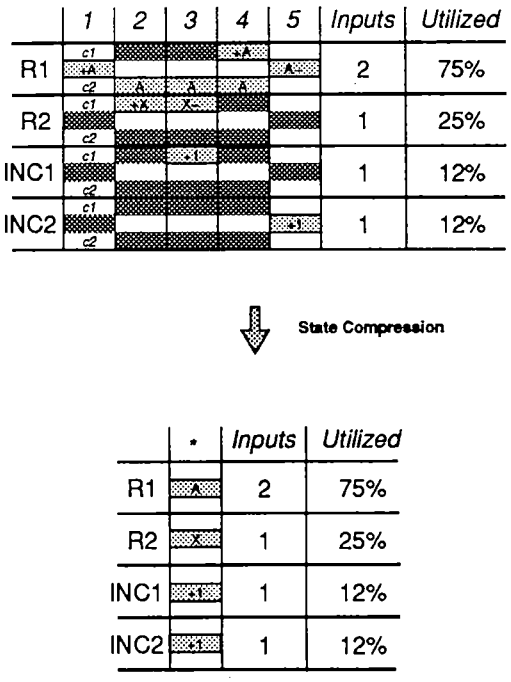



Figure 15: Unit–Assignment Display

### 5.2.4 Connection Allocation and Binding

The ability to specify unit connectivity, either partially or completely, and allocate wires or buses is also a necessary task for interactive design.

The floorplan display, besides allowing module and port placement, allows buses to be allocated and connections to be defined among the current set of components. The port–to–port connections shown in Figure 13 are an example of the most general kind of connection specification. Other kinds of connections, such as bus–to–port and port–to–bus connections, are specified by first allocating and placing a bus, and then specifying connections from module ports to the bus.

Partial connections are stored in the BIF connections list (described in Section 4.1.1). Newly allocated buses are stored in the unit list. The locations of wires and buses are maintained in the floorplan list.

The commands supported by the floorplan display for connection assignment are

- Bus allocation: a bus is added to a routing track. The width of the bus must be specified. Buses that will have more than one driver must be given unique names.
- Bus deletion: a bus is removed from a routing track. Any input or output connections are also removed.
- Connection assignment: a connection from an output port of a module to a bus, a connection from a bus to an input port, or a port–to–port connection is specified.
- Connection deletion: a connection on either side of a bus or a port–to–port connection is deleted.

### 5.2.5 Scheduling

Scheduling allows states to be split or combined to fulfill unit and timing constraints. Scheduling primitives are performed with the aid of XBIF, which also guarantees that the primitives are performed in the right order and do not alter the behavior of the design.

Unit constraints are taken to be the current set of allocated components. If a state requires more units than are available, XBIF highlights the state immediately.

Once unit constraints have been met, timing information is retrieved from the floorplan. XBIF uses this information to identify and highlight critical (longest) states. The designer may then perform further rescheduling.

Scheduling transformations are defined by the five primitives

1. state addition
2. state deletion
3. operation splitting
4. operation merging
5. operation relocation

State addition entails the adding of a new "empty" state which will have operations relocated to it. State deletion removes a state which has had all of its operations relocated to other states. Operation splitting divides an operation into two parts by selecting a subexpression within it and assigning its result to a temporary variable. The new operation is added before the original. The subexpression in the original operation is replaced with the new temporary variable. Operation merging does the converse by replacing a variable in an operation with the expression originally assigned to it. Operation relocation moves operations between states, above condition nodes, or below join nodes.

## 5.3 Relationships Between Design Views

This section describes an important feature of our environment: the ability to graphically display relationships between different design views.

Several user interfaces have been described already: XBIF, the resource utilization editor, and the floorplan display. In order to efficiently communicate to the user the relationships between the different abstractions of the design in progress, these displays must cooperate with each other to visually describe the correspondence between the different views represented. This requirement additionally necessitates that the central design representation maintain complete links between all stages of abstraction, and further, that these links be rapidly accessible by the environment's tools and displays.

Figure 16 shows an example of visual relationships between two displays. At the left, a BIF state was highlighted to request a display of the units performing the operation $result = result + opd$. At the right, the data path block within the floorplan display highlights the components and connections active for that operation.

We consider three design "entities" as candidates for graphical display of relationships.

1. states
2. units

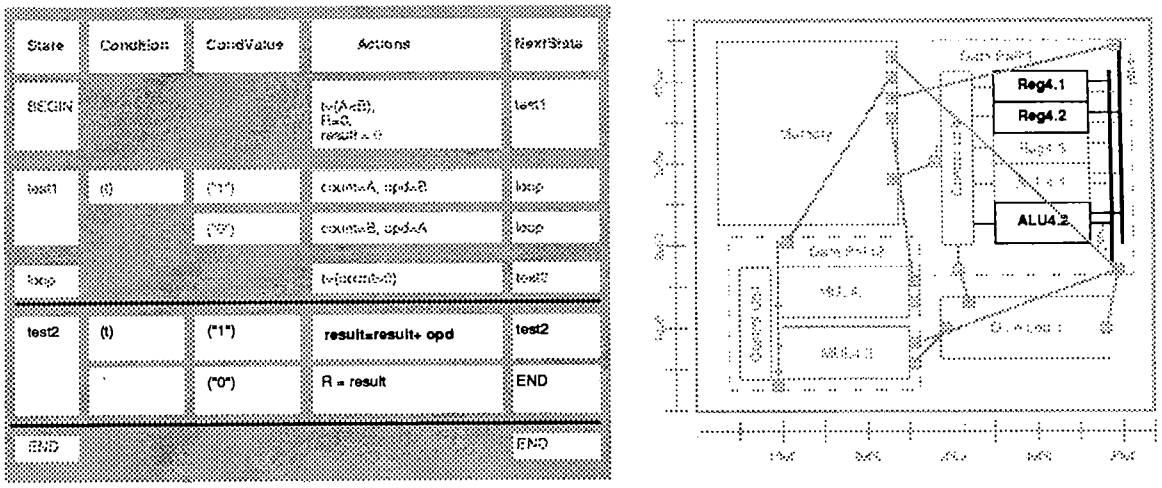



Figure 16: Relationships Displayed Visually

3. connections

States correspond to the rows in a BIF table. Units are design components. Connections are the wires between components. Relationships between these entities are expressed in the following two functions.

- State to units or connections: a selection of a state in XBIF highlights the corresponding units and connections in other displays.
- Unit or connection to states: a selection of a unit or connection in the schematic highlights the corresponding states in which the unit or connection is active.

# 6 Typical Scenarios In The Proposed Environment

This section describes some typical scenarios that might occur in the process of developing a design in our environment. Figure 17 shows a flowchart of the tasks that can be performed by the user with the aid of the displays. After each task, the user may continue with the next level of tasks, or, if necessary, return to an earlier one.

Initially, a design description is entered via XBIF to produce a structured BIF (SBIF) description. The complete description is then compiled to produce simulatable VHDL ([DuCH91]). In the process of compilation, undefined variables, unreached states, incorrect event values, etc., are identified and flagged. If the design passes through the compilation process without errors, the resulting VHDL can be simulated on a commercial VHDL simulator to verify the correctness of the specification.

Since the design behavior is entered in BIF, a state based language, the description always has an initial schedule. This schedule is achieved by allocating states for condition tests, loop points, join nodes, and straight-line code segments. Later, after unit allocation and assignment, the designer may discover that the schedule violates certain unit constraints. It will then be necessary to reschedule the design.

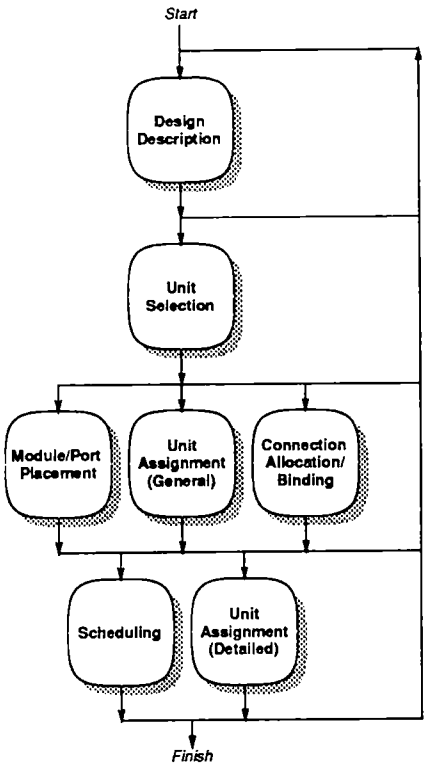



Figure 17: User Tasks Flowchart

Once the initial design description is entered, the next step will likely be unit selection. The component database is browsed via the unit selection display to identify and retrieve components that the designer wishes to become a part of the final design. This initial set of components is stored in the design description's unit list. At this point, the design may contain only a subset of the necessary components for implementation. Remaining components will eventually be allocated and assigned by the designer, or automatically by a module selector.

In many designs certain kinds of variables (index variables, accumulators) will be permanently assigned to unique storage elements. Likewise, the designer may wish to immediately bind certain operators (multiply, for instance) to special functional units. These kinds of initial bindings can be specified in the unit assignment display as soon as the desired unit has been selected with the unit selection display.

After unit selection, the designer can specify the initial placement for the components selected. (For bit-sliced components this task may be automatic.) Placement is accomplished by using the floorplan display. A rectangle corresponding to the width and height of the selected module may be placed in a position not overlapping any other module. Ports are displayed on the peripheries of the rectangle. As components are positioned, their locations, rotations, and port positions are recorded in the design description's floorplan list.

In addition to placement, an initial connection scheme may also be suggested by the designer. The floorplan display also allows buses or wires to be allocated and assigned to provide connections among the current set of components.

After connections have been partially or completely specified the designer may wish make further modifications to the floorplan scheme. Modules may be moved and rotated, and port locations on modules may be repositioned. The quality measures available from the floorplan display indicate

total area, wire length, channel density, and wasted area. Sub-tasks built in to the floorplan display give improvement hints to the designer by suggesting module placement and rotation, calculating centroid port locations for given nets, and suggesting optimal port positions for module groups. The designer can also remove a module, select a new module from the unit selection display and add it to the existing floorplan.

After unit selection and floorplan, the designer may find it necessary to reschedule the design. Using XBIF, the designer may split or combine states to achieve the desired unit constraints. XBIF can also highlight critical (longest) states if the units, connections, and floorplan of the design have been specified. This information allows the designer to address problem areas for timing constraints.

Since the initial schedule was entered in structured BIF, the designer may request that the representation be converted to unstructed BIF by collapsing operations states into condition states and removing joining states (shown in Figure 6). The resulting representation will more closely resemble the state to state behavior of the actual design implementation, but may not have easily identifiable control flow constructs.

As an alternative to manual scheduling, the design can be passed to an automatic scheduling tool. The designer may still make further modifications to the schedule to facilitate the unit assignment he or she has in mind.

Detailed unit assignment must be done at some point after unit selection. It is likely that both unit assignment and scheduling will be performed repeatedly by the designer until the desired unit and timing constraints are met. The resource utilization display lists the current set of allocated components and the states in which each unit can perform a storage or operational function. The designer then may specify the variable each allocated storage unit will contain in each state and the operator that each functional unit will perform in each state. If, after unit assignment, there remains unbound variables or operators, the designer has several options. Scheduling can be performed to increase the number of control steps and thus potentially free up units, or further units may be added. A third alternative is to pass the current description to an automatic allocation and binding tool that will attempt to complete it. The units allocated by the automatic tool will be added to the unit list.

After unit assignment and allocation is complete, any newly allocated components must be physically placed in the floorplan. Again, the designer or an automatic placement tool may perform this function. Once the floorplan is complete, the designer may now proceed to any of the previously mentioned tasks in order to optimize and refine the design. Now, however, the designer has a much better feel for the design's implementation, its floorplan, area, delay, and cost. With these in mind, he or she can select and replace units, modify connections, change the schedule, reassign variables or operators, or reposition ports and modules in the floorplan.

The links maintained between the behavior and floorplan of the design provide direct visual feedback information between XBIF and the floorplan display, assisting the designer's decision making process.

# 7 Summary

We have presented a framework for a decision support environment for behavioral synthesis. The environment fully supports user control at every level of design abstraction, allowing maximum utilization of designer experience. Rapid feedback of physical design information conveys to the user the proper goal-oriented information he or she needs to explore the design space efficiently. The design representation model preserves relationships between behavior and structure and allows the user interfaces to graphically display this information directly to the user.

# 8 References


[ChGa89] Chen, G.D, and Gajski, D.D., "An Intelligent Component Database System for Behavioral Synthesis", University of California, Irvine, Technical Report 89-39, 1989.

[WuCG91a] Wu, A.C-H., Chaiyakul, V., and Gajski, D.D., "Back-Annotation for Interactive Data Path Synthesis", University of California, Irvine, Technical report 91-29, April 1991.

[WuCG91b] Wu, A.C-H., Chaiyakul, V., and Gajski, D.D., "Layout Area Models for High Level Synthesis", University of California, Irvine, Technical report 91-31, April 1991.

[DuCH91] Dutt, N., Cho, J., and Hadley, T., "A User Interface for VHDL Behavioral Modeling," CHDL, April 1991.

[Dutt88] Dutt, N., "GENUS: A Generic Component Library for High Level Synthesis", *Tech Rep 88-22*, UC Irvine, Sept. 1988.

[DuHG90] Dutt, N., Hadley, T., and Gajski, D., "An Intermediate Representation for Behavioral Synthesis," *27th Design Automation Conference*, June, 1990.

[RuGa90] Rundensteiner, E. A., & Gajski, D. D., "A Design Representation for High-Level Synthesis", Info. & Computer Science Dept., UCI, Tech. Rep. 90-27, Sep. 1990.

[RuGa91] Rundensteiner, E. A., & Gajski, D. D., "BDEF: The Behavioral Design Data Exchange Format" Info. & Computer Science Dept., UCI, Tech. Rep. 91-34, April 1991.

[Sh89] Shung, C., et al., "An Integrated CAD System for Algorithm-Specific IC Design," Proc. Int'l Conf. System Design, Jan., 1989.

[DeRa86] DeMan, H., Rabaey, J., et al., "Cathedral II: A Silicon Compiler for Digital Signal Processing," IEEE Design and Test, Dec., 1986.

[BrCa88] Brayton, R., Camposano, R., et al., "The Yorktown Silicon Compiler," *Silicon Compilation*, Addison Wesley, 1988.

[CaWo91] Camposano, R., Wolf, B., *High Level Synthesis of VLSI Systems*, Kluwer, 1991.

[PaPM86] Parker, A., Pizarro, J., and Mlinar, M., "MAHA: a Program for Datapath Synthesis," *23rd Design Automation Conference*, 1986.

[THKR83] Thomas, D., Hitchcock, C., Kowalski, T., Rajan, J., Walker, R., "Automatic Data Path Synthesis," *IEEE Computer*, December, 1983.

[PaGa87] Pangrle, B., and Gajski, G., "Slicer: A State Synthesizer for Intelligent Silicon Compilation," *International Conference on Computer Design*, 1987.

[TsSi83] Tseng, C., and Siewiorek, D., "Facet: A Procedure for the Automated Synthesis of Digital Systems," *20th Design Automation Conference*, 1983.

[DeNe89] Devadas, S., and Newton, R., "Algorithms for Hardware Allocation in Data Path Synthesis," *IEEE Transactions on Computer-Aided Design*, Vol CAD-8-7., July, 1989.