Report 83-16
Stanford -- KSL

Scientific DataLink

Behavioral Specification and Verification within a VLSI Design System.
Gordon Foyster,
Apr 1983

Heuristic Programming Project
Report No. HPP-83-16

April 1983

Behavioral Specification and Verification within a VLSI Design System

GORDON FOYSTER
HEURISTIC PROGRAMMING PROJECT
DEPARTMENT OF COMPUTER SCIENCE
STANFORD UNIVERSITY

This research is funded by the Defense Advanced Research Projects Agency under Contract MDA-903-80-c-007.

Behavioral Specification and Verification within a VLSI Design System

Gordon Foyster
Department of Computer Science, Stanford University

Abstract. *Palladio* is an experimental system for designing very large scale integrated (VLSI) circuits. A design is entered by using components viewed from one or more *perspectives*. Each perspective provides a way of looking at the *structural* composition of a component.

Simulation is currently the fastest and most widely used method of establishing the validity of circuit designs. In order to simulate a circuit we must be able to model the behavior of components used in the circuit. A user specification of behavior can be compared via simulation with the behavior of the component as a composite of more primitive components. By assigning behavior to higher level components we usually obtain large speed increases in the simulation.

A method is given for stating the behavior of components used within Palladio and circuit verification through simulation.

Introduction

The Palladio system is designed to provide intelligent assistance to all stages of design. Interactive graphics and menus are the main forms of interface with the user.

In keeping with this philosophy of interactive assistance at all stages of design, it was necessary to develop a method for verifying a design that a) was not fully specified, and b) possibly used components viewed from different perspectives or *abstraction levels*. Designers need to be able to check parts of designs and designs that are still very abstract. In order to do this they must have a means of saying what the design or partial design is supposed to do with given inputs, and then run a program which can verify this behavioral specification. The behavioral specification and verification needs to be fully integrated with the rest of Palladio to allow easy transition between the structural construction/modification mode to the testing/verification mode. In order to avoid translation from the structures constructed by the Palladio editor to structures used by verification through simulation it is necessary for the simulator to directly reference the Palladio composed structures.

This document is divided into three parts. The first details the requirements for a system to provide an interactive environment for behavioral specification and verification, with an outline of the solutions for those requirements. The next section is a user manual for behavioral specification and simulation. The last section deals with the issues involved in implementing these facilities and integrating them with the rest of Palladio.

1. Requirements

1.1 Conceptual Framework

There are two main ideas used in behavioral specification:

1) A behavior can be assigned to a structural component which is independent of its environment. In this way library components have prototypical behavior which is inherited by all instances of the library component used in the circuit. The major difficulty with behavioral specification is that the environmental independence requirement often does not seem natural. The main aim of the system to be described is to provide a method which allows behavioral specification for components in isolation, yet still produces expected behavior when these components are structurally connected.

2) The behavior of an artifact is, in general, its response over time to changes in its attributes. For a circuit component these attributes are the values of signals on the ports of the component and the internal state of the component. The behavior of an electronic device is then given by describing the signals appearing on the output ports over time, given changes on the input ports. For different perspectives there may be different attributes which affect the behavior of a device. Eg the contents of a register for structures viewed from a register transfer perspective, or the resistance to ground when viewed from layout.

Simulation means modeling the attributes of structures in the computer, specifying a set of initial conditions for attributes in terms of this model, then applying the behavioral specifications of all affected components in order to produce a further set of attribute values for times in the future.

1.2 Representation Scheme

The representation scheme used within Palladio is LOOPS[4]. This is an object and data-oriented programming system for use in Interlisp. LOOPS objects are record structures with slots for storing attributes of the object. Objects can respond to messages. This is implemented by associating an Interlisp function with each message. Slots can have *ActiveValues* which cause functions to be called when the slot is read or written.

Circuit components are LOOPS objects and attributes of components are stored in slots given by the name of the attribute. Messages sent to objects are used for component display and editing of structure and behavior. The use of LOOPS code objects to represent circuit objects produces a very close analogy between the code structure and the circuit structure.

1.3 Behavior Specification

There a four reasons for specifying behavior for Palladio components:

1) The behavioral specification allows a simulation to closely model the behavior of the components in a real circuit.

2) To compare the expected behavior of a composite object with the behavior derived from its sub-components.

3) The functionality of a component at one level of abstraction may influence how it is transformed to another level of abstraction.

4) A component can be used as a "black box" to generate signals to be used in simulation. This component might not have an analog in any real device.

Behavioral specifications in Palladio take the form of rules which tie a statement about an attribute that has changed to another attribute of the component. The rule is an implication:

if *changed attribute* then *new attribute*

In our system there is no attempt to capture the designer's intent for the complete behavior of the component. The component will only be able to react to events which are specified on the left hand side of a behavioral rule. This contrasts with work where the designer gives a *specification* of what the outputs *should* be [7].

1.4 Simulation

The simulator is based upon MARS[5], a Multiple Abstraction Rule-based Simulator. By *forward chaining*, using the logical inference rule *modus ponens*, on the behavioral rules from a set of initial conditions, new states can be found. The new state can then be used recursively for forward chaining. MARS uses the Metalevel Representation System (MRS) [3] for its inferencing and state representation. The simulator mechanism of MARS is domain independent and could be used to produce an event-driven simulation of any system. However, within Palladio the domain is electronic circuits and states are determined by values on ports of components and stored state within components eg. registers.

The user needs to be able to specify the values to be recorded and how. Menus and graphical interaction allow selection of which ports of a component are to be set to initial conditions and those whose simulated values are to be recorded. Control over all stages of the simulation

output is allowed through user access to every statement about a changed attribute. This allows the user to define routines to change the graphics in response to changed attributes, creating animation effects during simulation.

Behavioral connection between components is provided by ports. All ports have a record of the component of which they are a part and the wires, if any, which connect them to other components. If the component has sub-structure then a port may also be connected to a wire which is part of the sub-structure. When the state of a port changes then the simulator forward chains on the rules associated with the externally connected wire along with:

a) rules on the port's component if the top level behavior of the component is being used.

b) rules on the internally connected wire if the simulation is being run on sub-structure.

c) both (a) and (b) if the top level behavior is being *verified* by simulation using sub-structure. In this way the user can compare a gross behavioral specification of a component with the behavior found by simulation of the sub-structure of the component.

1.5 Reasoning in an Object-Oriented System

Because the simulator uses the MRS inferencing system, and the data is represented in an object-oriented system, there is a need to combine these systems and allow them to work together. This is done by defining a strict interface between the MRS inferencing mechanism and LOOPS.

The interface protocol is:

(i) MRS uses slots on objects to store propositions about those objects.

Eg. The proposition:

(John height 72)

uses the "height" slot of object "John" to store the value 72.

(ii) In order to combine the default representation scheme used by MRS with the capability of objects, a tag is needed for those cases when the LOOPS objects are to be used for storage or retrieval. In our system a "LOOPS" tag is added to propositions which refer to LOOPS objects. This "LOOPS" tag is invisible to the user. The meta-level capability of MRS is used in invoke the LOOPS frame representation for storage and retrieval of LOOPS propositions.

(iii) Implications which affect a particular object are stored with that object. This decreases the work needed to access the object's rules for inferencing or editing.

LOOPS provides slots with an *ActiveValue* capability which allows user-defined routines to

be called whenever object slots are referenced or changed. This provides a powerful "demon" mechanism for performing side-effects to the reasoning. A side-effect might be the display of attributes which have changed.

MRS is used mainly for its meta-level capability, although there is very little need for reasoning at the meta-level since the LOOPS tag makes the storage and retrieval method explicit. The other part of MRS used by the system is the unifier which is used to match the left hand sides of rules for forward chaining..

By combining LOOPS and MRS in this way we can utilise the rapid lookup of facts and storage economy of a frame based representation system along with the flexible application of added knowledge of a logic programming system. The inferencing rules are able to refer directly to the state of the system modeled by the objects.

1.6 Signal Types

It is anticipated that most behavioral specifications will be in terms of a Signal attribute associated with the ports of a component. There exist several defined signal types for different forms of functional specification and the user is free to create another form of signal if desired.

The existing signal types are:

LOGIC - Three values: HIGH, LOW and UNDEF.

ARITH - Signals are integers with arithmetic operations generally used to determine outputs from inputs.

SIGSTR - Each signal has an integer strength and level. The strength corresponds to the driving force of the signal. This depends on the capacitance for stored signals and the resistance to ground of driven signals. The level is the voltage of the signal. Signals also have an origin so that functional rules can determine whether to over-write an existing signal, based on who is driving the signal. This signal type is needed for specifying behavior for bi-directional components and circuits where resistive and capacitive effects are necessary. It is the major mechanism allowing specification of the behavior of components independent of their environment.

Eg. If two wires were providing inputs to a contact, with one wire going out, then if the two input signals had different levels we need to decide on the signal level of the output. When we simulate this system we make the contact point a Palladio component and give it a Signal attribute which is the signal to be propogated to the output. If one input has already changed the Signal of

the contact then when the second input tries to change the contact's Signal it notices that the signal level is different from what is already there. If the source of the existing signal was the same as the new signal then the new signal can overwrite the Signal of the contact. Also, if the strength of the new signal was greater, then the new signal will over-write the previous signal.

A predicate called OverWrites, which takes two SIGSTR signals, exists for use in functional specifications. This predicate is true if the first signal overwrites the second signal. This signal type is based on the work of R. Bryant[2] with extensions by N. Singh.

A circuit can be simulated using a number of signal types at once. For example, the designer might wish to propogate the signals to an adder as intergers but in order to check the design of the adder it must be simulated using boolean logic signals. Control over simulation stepping, tracing and break conditions similar to a software debugging environment is also available. If the simulation uncovers an error in the design then the user can change the structural or behavioral specification of the design. By simulating a circuit using both its specified behavior and the behavior implied by its sub-component structure the user can obtain some automatic verification of the correctness of the circuit. The completeness of this verification depends upon the completeness of the input vectors for the simulation.

1.7 Need for Knowledge

The system must possess knowledge in order to provide the following features:

1) Simulation of connected components should agree with the designer's intuition for the behavior of connected components.
2) The simulator should use the correct signal type for components being simulated.
3) Components should be simulated using the correct perspective.
4) Control of the simulator should be flexible and easily modified.

Some of this knowledge is provided by rules such as those controlling simulation output. These rules can be changed by a user of the simulator. The behavior rules allow different signal types to be intermixed by including predicates on the signal type in the left hand side of the rule. Other knowledge is provided by the system builder in the form of modular functions. These functions know about connections of wires to ports to propogate signals during simulation. (The functions could be rules which are interpreted for each new state, at a high cost in speed of simulation.)

1.8 Further Work

1) Rule compilation: Many of the behavioral rules and simulation commands must determine the truth of predicates which remain fixed throughout the simulation. Currently this truth determination is done at execution time by interpreting the rule or precondition in order to determine its truth. It would greatly speed up the simulation to have an initial intelligent system to compile the rules. The compiler would look at all rules, determine those whose status was fixed and replace them by the interpreted value once only, before the simulation.

2) Using behavior rules for more than simulation: Such things as transforming from one structural perspective to another, diagnosing faults in components and generation of suitable test vectors for simulation would all be aided by knowing the component's behavior. The rule format for behavior specification makes it particularly suitable for further applications.

Example of behavioral specification and verification.

When the Palladio user enters the simulator he is shown a picture of the circuit as in Figure 1 and presented with a menu of commands.

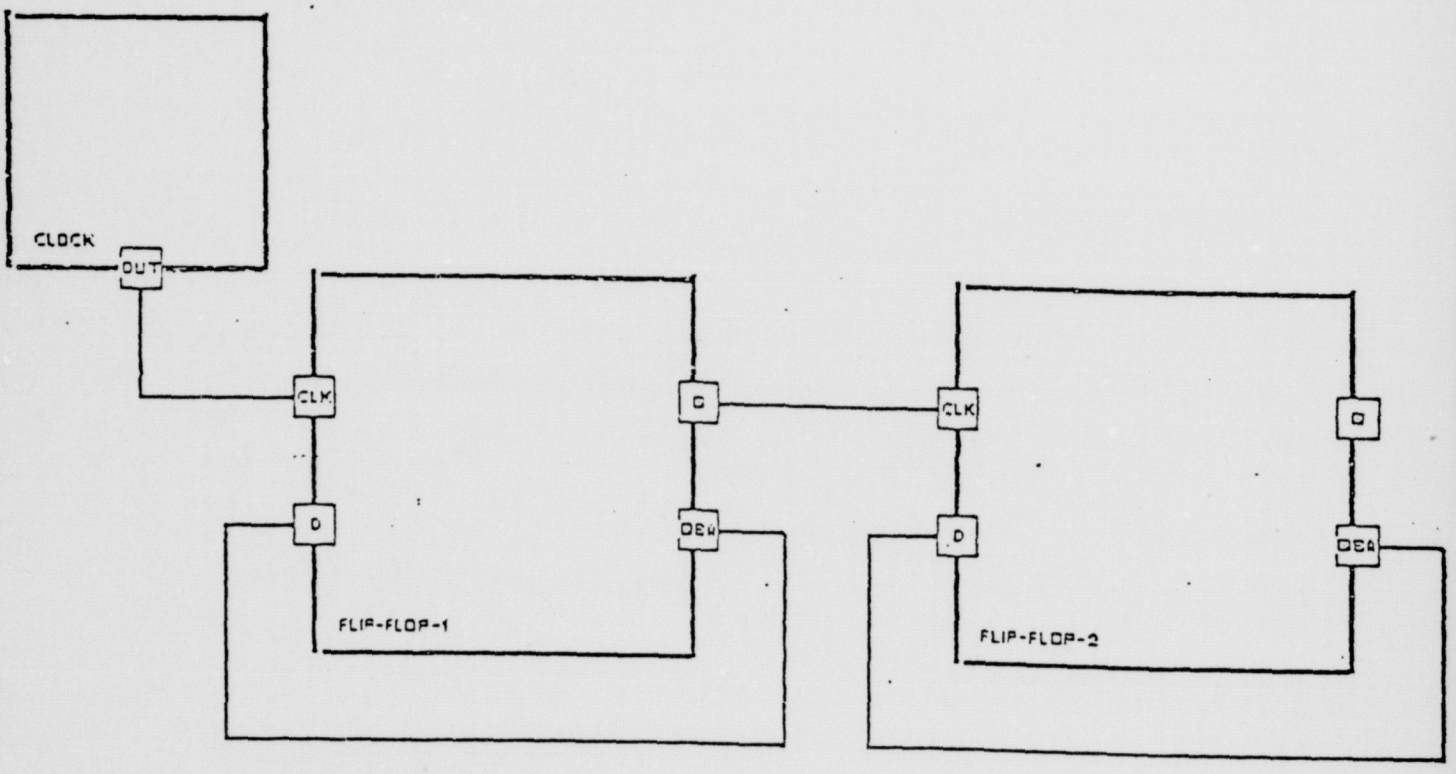

Figure 1. Diagram of circuit to be simulated

Simulation Menu Commands:

SetPortSignal, Run, SingleStep, Trace, SetBreaks, SetTime, Reset, SetComponentValues, ShowComponent, EditSimulatorRules, VerifyComponent, EditComponentBehavior, EditCircuitBehavior, EditGlobalBehavior, EditStructure

The first 8 commands are concerned with running the simulator. SetComponentValues is used to change such things as capacitance of a wire or delay of an inverter. VerifyComponent allows a component to be selected so that it is simulated using both its high level description and sub-component structure and the results are automatically compared. The Edit~Behavior commands allow the behavioral description of the circuit or sub-components to be changed before or during a simulation. EditStructure allows the Palladio's structural editor to be used on the circuit.

If EditComponentBehavior is selected, followed by selecting FLIP-FLOP-1 from the displayed diagram, a screen text editor is entered with the behavioral rules associated with that component.

An example Behavioral Rule:

```
(if (TRUE (Value (Pin CLK) Signal HIGH) $T)
 then
  (and (TRUE (Value (Pin Q) Signal (Value (Pin D) Signal)) (ADD1 $T))
  (TRUE (Value (Pin QBA) Signal (INVERT (Value (Pin D) Signal)) (ADD1 $T))))
```

This gives the behavior of the flip-flop with a delay of 1 between the CLK going high and the D input appearing on the Q output, and the inverse of D appearing on Q-bar. Names starting with "$" denote variables which can be bound to any value. The syntax of behavioral descriptions is given in the section on behavioral specification.

Initial conditions are specified by selecting SetPortSignal. The port, signal type, signal value and time for the signal to be set can then be entered.

By selecting SingleStep or Run from the menu we start the simulation. SingleStep moves to the next time for which events are happening and runs only those events at that time. Run causes the simulation to run until termination or a break condition occurs. The output of the simulation is determined by simulation rules such as:

```
Eg. (if (TRUE (Value $OB Signal $S) $T)
        then (SEND $OB Redisplay (Value Simulator SimViewport)))
```

This would give an animated graphic effect if the Redisplay message to a pin caused different Signal values to be shown differently.

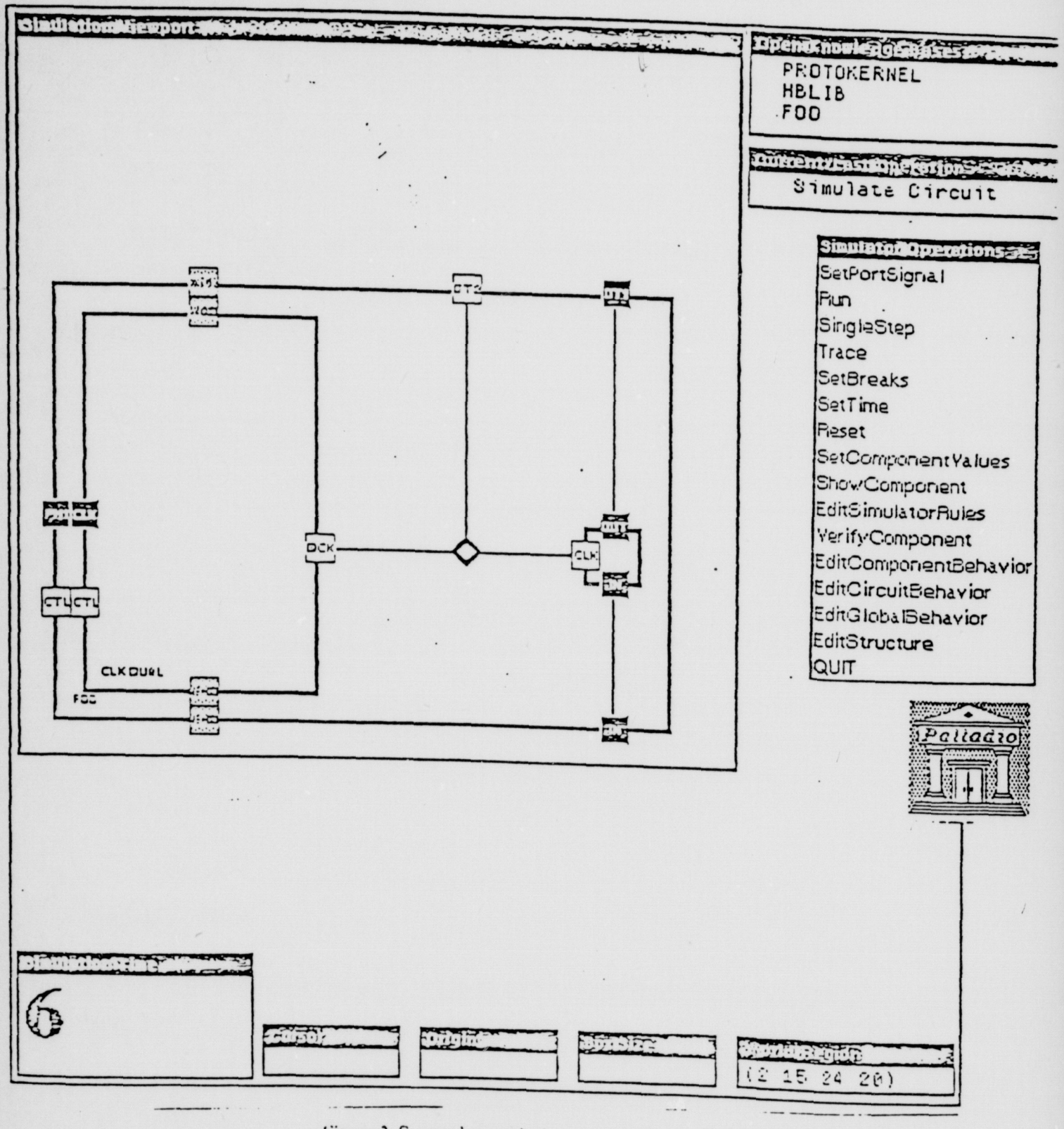

Figure 2. Screen layout during simulation

2. User Manual

When the Palladio user enters the Simulate Circuit section from the top level he is presented with a graphical view of the circuit and a menu of commands. Figure 2 shows a picture of the screen within the circuit simulator. As well as the menu of commands shown, there is a pop-up menu of commands for altering the graphical view of the circuit. The following sections detail how the commands are used.

2.1. Behavioral Specification.

The default behavior of library components provided by Palladio is specified as the library is being built or modified. A user is able to override this default behavioral specification by first selecting *EditComponentBehavior* from the command menu, then selecting the component to be modified and editing the behavioral rules using a screen text editor. If the behavior of the whole circuit is to be modified then the user first selects *EditCircuitBehavior* from the menu then uses the text editor as before. A component can have many behavioral rules. Each rule has the following syntax.

Behavioral Rule Syntax.

In this section we describe the syntax for behavioral rules followed by examples for most of the cases.

Behavioral rule ::=

(if <LHS> then <RHS>)

<LHS> ::=

<time statement>

or (AND <time statement> <condition> <condition> . . .)

or (OR <LHS> <LHS> <LHS> . . .)

<time statement> ::=

(TRUE <condition> <variable>)
or (TRUE <condition> <number>)

<condition> ::=
<value statement>
or <predicate>

<value statement> ::=
(i) (Value <attribute chain> <value>)
(ii) or (Value <attribute chain> <variable>)
(iii) or (Value <attribute chain>)
(iv) or (Value <variable> <attribute chain> <value>)
(v) or (Value <variable> <attribute chain> <variable>)
(vi) or (Value <variable> <attribute chain>)

<attribute chain> ::=
<attribute>
or <attribute> <attribute> <attribute> . . .

<attribute> ::=
slot of previous element in chain or current component
if first slot
or (slot of previous element object name)

<value> ::=
<constant>
or <behavioral function>

<variable> ::=
a character string beginning with $

<behavioral function> ::=
LISP function taking constant parameters or value statements
of type (iii) or (vi)

PAGES MISSING FROM ONLY EXTANT AND AVAILABLE COPY

BLANK PAGE

To test for different forms of signal specification:

```
(if (and (TRUE (Value (Pin IN) Signal $S) $T)
         (SigType? $S ARITH))
  then
    (TRUE (Value (Subcomp FOO) (Pin IN) Signal (BIT-1 $S)) $T))
```

Using a variable in the first position for a value statement to define a GlobalRule for when an object is active:

```
(if (TRUE (Value $OB Signal $S) $T)
  then
    (TRUE (Value $OB Active? YES) $T))
```

2.2. Simulation

In order to simulate the circuit the user must provide a set of initial conditions and run control. Simulation control is provided by means of user-defined break conditions and stepping controls. Output is determined by selecting components to be traced and defining different output modes. Eg. A graphical trace of the Signal on a pin provides a "cartooning" effect on the circuit. Other output such as timing diagrams and writing values to a file can also be specified. The user is given an interface which allows a new output function to be easily integrated with the simulator.

The simulator is event driven as opposed to the propogation of state values. This means that behavioral rules are only used to determine new values for attributes when an event has occurred which might have an effect. That is, there is no concept of signal decay or change in any way outside that expressed explicitly by the behavior rules.

Run Control

The simulator can be run from the current time to termination or a break condition by selecting the *Run* command. The *SingleStep* command executes those events at the next time step. The next time might not be the current time incremented by one but depends on when the next set of events are to happen.

Tracing and break control are provided by the *Trace* and *SetBreaks* commands. Trace allows selection of ports or areas of the circuit to be traced and break conditions allow the user to stop the simulation when the condition occurs.

Selection of level of simulation

Since components within Palladio circuits can have substructure and the behavior of components can be specified at any level, then the simulation could be performed at any of the structure levels. The default is to use the highest level of structural description for which the behavior has been specified. If the behavior has not been specified for any level of structure then that component cannot be simulated and signals will not be propogated once they arrive on the input pins.

The user can choose between using a higher level description, usually resulting in a faster simulation, or the sub-structure where the internals of a component are to be simulated.

Note that the level of simulation may not correspond to the level of structure at which the circuit is viewed. Successive levels of component sub-structure are viewed by using the ShowComponent simulation command and then selecting the component whose internal structure is to be shown. This causes a window to be created showing the details of that component only.

Changing output functions

By selecting EditSimulatorRules from the menu the user can edit the commands to the simulator. These commands are also rules but not implications since the r.h.s. is a LISP function to be executed if the l.h.s. is true. The user can reference attributes associated with the simulator by using the Value term as a function with the first argument being Simulator.

The syntax of SimulatorRules is:

<SimulatorRule> ::=
(if <L.H.S.> then <behavioral function>)

Eg. (if (TRUE (Value $OB Signal $S) $T)
then (SEND $OB Redisplay (Value Simulator SimViewport)))

The SEND function is used within the object-oriented programming system upon which Palladio is built.

Multiple dimensions for simulation.

There are three dimensions across which the Palladio simulator works:

1) Component hierarchy - each component can be expressed in terms of its sub-components at one structural abstraction of Palladio.

2) Structural perspectives - these are fundamentally different ways of looking at circuits and components. Different abstractions deal with different aspects of the design. Different perspectives are defined by assigning different attributes to components. Within Palladio the perspective hierarchy usually follows the component hierarchy. That is, a component specified at a high level may have different attributes to its sub-components. Eg. A flip-flop has a "state" attribute which is not possessed by the component gates.

3) Signal types - different signal types used implicitly in functional specifications must be handled by the simulator.

Earlier sections have have shown how to handle the first hierarchy within simulation. In order for a circuit specified at different perspectives, and with different signal types to be simulated, the user must have specific procedures for transferring between levels within the behavior rules. This means that the user must directly reference those attributes relevant to a perspective when the behavior of a component at that level of the component hierarchy is being specified. He must also incorporate the functions for changing between signal types within the behavior rules, for components using more than one signal type. The SigType? predicate can be used as a precondition to allow signal types to be determined when the rule is being interpreted during simulation.

2.3. Verification

In this system Verification means comparing the simulation outputs of a component for which both the top-level and sub-structural descriptions of behavior are to be used. Select VerifyComponent from the menu then select the component which is to be verified. When an inconsistency occurs the simulator asserts a proposition of the form:

(TRUE (VerifyError *object attribute high-level-value low-level-value*) *time)*

This proposition can be trapped by the simulator rules to produce user defined results. The default is:

```
(if (TRUE (VerifyError $OB Signal $Val1 $Val2) $T)
  then (SEND $OB DisplayError $Val1 $Val2 (Value Simulator SimViewport)))
```

2.4. Simulator Command Menu Summary

SetPortSignal Allows signals on ports to be set for any desired time.

Run Run the simulation, stopping at termination or break condition.

SingleStep Execute the events for the next time in the event queue.

Trace Allows selection of ports to be traced.

SetBreaks Allows specification of break conditions to halt the simulation.

SetTime Set the current simulator event time.

Reset Set the simulation time to zero and all ports to undefined.

SetComponentValues Allows attributes of components to be set.

ShowComponent A selected component will have its internal composition desplayed in a new viewport.

EditSimulatorRules Allows the rules controlling simulation output and side effects to be changed.

VerifyComponent Performs a comparison of simulation using both the top level behavioral specification and the behavior implied by the subcomponents for the selected component.

EditComponentBehavior Change the behavior of a selected component.

EditCircuitBehavior Change the behavior of the top level circuit.

EditGlobalBehavior Edit rules which apply to all components.

EditStructure Transfer to the structure editor for circuit changes.

QUIT Return to the top level of Palladio.

3. Implementation

System Configuration

Figure 3 shows the necessary systems for simulation of circuits within Palladio. HILGA[6] is a High-Level Graphics package based upon LOOPS, where graphics objects are LOOPS objects.

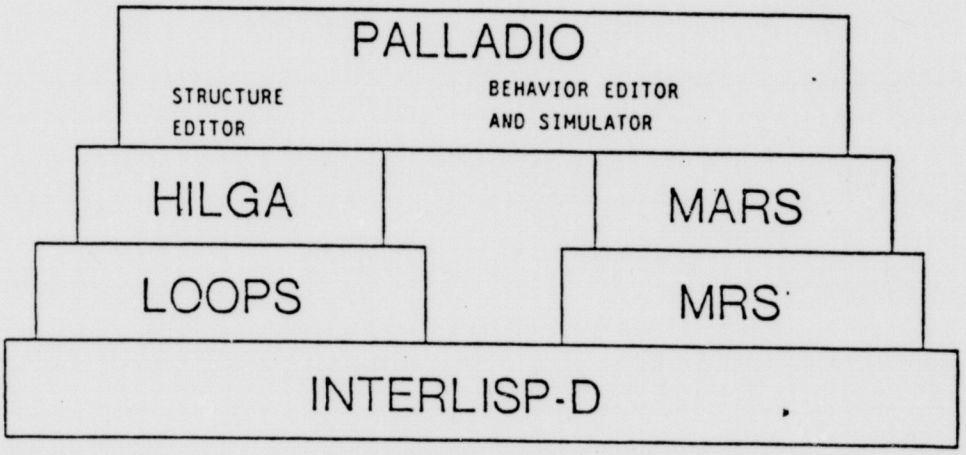

Figure 3. Systems required by Palladio simulator

The following features provided by LOOPS are used by the Palladio simulator.

1) Object-oriented programming.

Objects are frame structures with slots.

Objects respond to messages by performing an associated method.

2) Classes which are templates from which instances can be made.

3) Classes can inherit methods and slots from multiple super classes.

4) Slots can be Active Values which cause functions to be run whenever the slot is referenced or changed.

These features are used as follows:

Any class for which the functionality of instances is to be specified must be a sub-class of the class BehaviorObject. All circuit components and sub-components are instances of classes with this super-class. BehaviorObject has a slot SpecifiedBehavior which is a pointer to an instance of the BehaviorRules class. Figure 4 shows the relationships for a typical component class used within Palladio. When the user edits the behavior of a component he is changing a slot called UserForm

on BehaviorRules. This slot value is translated upon change into a form to be used when simulating and stored in the BehaviorRules slot SimulatorForm.

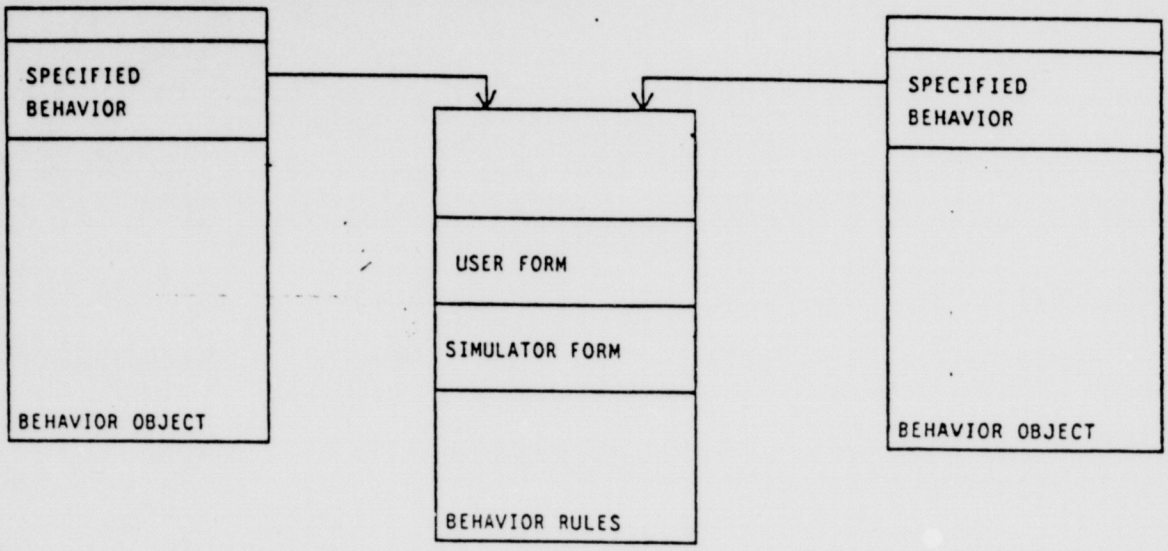

Figure 4. Class relations for Palladio components.

Fielding menu commands.

When a simulation command has been selected from the the menu it is sent to a *Simulator* object. The Simulator object has slots which are used by methods for performing the commands.

The most important slots on the Simulator are:

CircuitOb : A pointer to the current circuit object being simulated. Subcomponents of the circuit are accessed through this object.

SimWorld : The HILGA *GRAPHICSWORLD* containing the circuit for viewing purposes.

SimulatorRules : Contains an instance of BehaviorObject which allows the Simulator to have behavior.

CurrentTime : The current simulation time.

Running a simulation.

MARS provides the mechanism for sequencing of events. An event is a proposition about the state of the circuit a given event time. The propositions used when simulating Palladio circuits are of the form:

(TRUE (LOOPS Value *object slot value*) *time*)

When a proposition of this form is asserted as being true the LOOPS tag is recognized by a metalevel trap facility provided within MRS and the proposition is passed to the simulator system interface of Palladio. This interface looks at the SimulatorForm of the SpecifiedBehavior of *object* and if the proposition matches the l.h.s. of any rules then the r.h.s. is asserted. ie. a new state is determined by *forward chaining* from the current state using those rules associated directly with the object being referenced. The metalevel facility of MRS recognises assertions of the form (TRUE (*proposition*) *time*) to be passed to MARS. If the time is the current time then this new proposition is asserted directly - to pass back to the Palladio interface. Otherwise *time* will be in the future and MARS stores the proposition on a heap for later assertion. MARS also provides a mechanism for checking propositions for a given time to ensure that the same proposition is only asserted once.

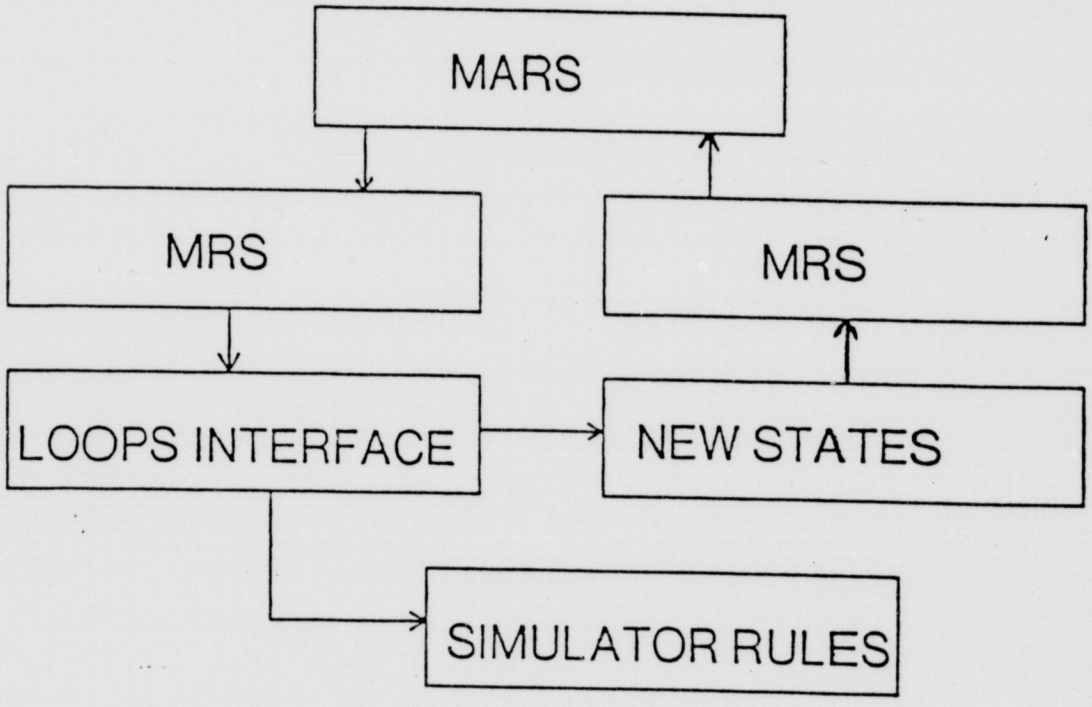

Figure 5. System interface

The above description glosses over a few facts. Ports are objects but do not have behavior, only serving to connect components with the outside world. So the Port class is not a sub-class of BehaviorObject. When a proposition is asserted where the *object* is a port then the simulator interface applys the rules on the externally connected wire and either or both of the port's component and the internally connected wire, depending on the simulation mode for the component. When only the top level behavior is to be used then only the component's behavior