VIDE: A Visual VHDL Integrated Design Environment*

Jinian Bian, Hongxi Xue and Ming Su

Dept. of Computer Science and Technology
Tsinghua University
Beijing 100084, CHINA
Tel: +86-10-62785564
Fax: +86-10-62562463
email: [bianjn|xuehx|suming]@tiger.cs.tsinghua.edu.cn

Abstract– In this paper, a visual VHDL integrated design environment VIDE for high level design is presented. In VIDE, there are several graphical and textual mixed design entry tools (VDES) and a graphical object-oriented debugger (VDBG). VDES consists of several diagram editors and a visual text editor, while VDBG is a debugging environment based on a hierarchical VHDL simulator. The graphical objects can be specified as a debugging target.

I. INTRODUCTION

VHDL has strong description facilities. However, the complexity of VHDL syntax and flexibility of the description make it very difficult for the user to code and understand the description programs. Since IC design becomes larger and more complex rapidly, traditional VHDL entry approach is more and more difficult for designers and is easier to cause some mistakes.

To solve the problems, a convenient graphical design entry approach and a convenient verification debugging tool will make it much easier for the user to enter his design and find his mistakes. Since most designers are familiar with graphical approach to describe their design idea, to build a graphical design entry environment is useful. Another way is to build an interactive simulator with convenient visual debugging.

In this paper, a graphical object-oriented visual VHDL integrated design environment (VIDE) is presented. We propose an object-oriented graphical model for design entry approach and a graphical object-oriented visual debugger based on a hierarchical simulation.

II. OBJECT-ORIENTED GRAPHICAL DESIGN ENTRY TOOLS

A. The Architecture of the Graphical Design Entry Tools

VIDE consists of several object-oriented visual design entry tools (VDES) and a visual debugging environment based on a hierarchical mixed-level VHDL simulator (VDBG). Using VIDE, the designer can enter and verify his design conveniently. It can also support synthesis tools. The whole environment is managed through design library manager.

The hardware designers prefer diagram approach to show their design ideas and design results. VDES supports several graphical editors and a visual text editor (TXT). The graphical editors are for finite state machine diagram (FSM), data flow diagram (DFD), control flow diagram (CFD), structural schematic diagram (SCD). The diagrams and the text are hierarchically mixed together to form a complete design.

Each graphical entry tool consists of 3 parts: an editor, a transformer and a translator.

Each editor is an interactive user-interface. The designer can enter, load and edit his design in a graphical manner or/and a textural manner. The logic data of the result are translated into VHDL or other hardware description languages such as Verilog by the translator.

A transformer is used to isolate the editor and the hardware description language (HDL), so the editor orients objects. The graphical diagrams are independent of the HDL, so the designers need not be familiar with the HDL in detail. The transformers also do some rule-checking of the graphical diagram.

* This work is supported by National Project and Tsinghua Science Research Foundation.

ASP-DAC '97
0-89791-851-7$5.00

B. *The Object-Oriented Graphical Models*

1) Graphical Objects

In all of the editors, besides the graphical information, there are some information which can not be displayed. So a graphical model consists of graphical objects and textual objects.

A textual object is unable to display in a graphical window, such as a data type, a package, the synchronous or asynchronous flag.

A graphical object is a principal object or an attribute object. A principal graphical object is a main graphical element of the diagram, such as block, state node, signal, port.

An attribute object is a displayable string, such as a name, a parameter, an electrical value and a notation, which is subordinated to a principal object. An attribute object can be operated independently. The user can create, delete, move, and change its position and its size. If a principal object is deleted, then all of its subordinate attribute objects will be deleted automatically.

In the data flow diagram (DFD), a principal graphical object can be a *block*, a *signal* or a *port*. A block is displayed as a rectangle or a circle, which stands for a functional block, a process or a submodule. They are parallel. The communication between two blocks is through a signal. Each signal is a broken line with two ends (ports), which connects two blocks and stands for a data flow. The signal has a value of a data type, and it can have an initial value. A signal port must be specified in mode *in*, *out*, or *inout*, which decides the direction of the data flow.

A control flow diagram (CFD) expresses the internal sequential program of a block in graphical approach. There are 3 kinds of graphical symbols: *control node*, *control flow* and *local connector*. A control node has 7 types: *start*, *end*, *action*, *decision*, *wait*, *loop-start* or *loop-end*. An action node is a functional module. A wait node is correspondent to the *wait* statement in VHDL. The other nodes express control sequences. A control flow consists of one or more line segments. A connector is used to break the line of control flow. It is an *entry connector* or a *leave connector*, which is connected by the same name. Each graphical node can have its name and its attributes. The action node should be specified as the action statements like HDL statements. A finite state machine diagram expresses a finite state machine, that is, the internal sequential process of a functional element by transition.

In a finite state machine (FSM) model, there are 4 kinds of graphical symbols: *state*, *transition*, *local connector*, *global connector*. A FSM may be a Moore or Mealy model. A state is shown as a circle. Any two state nodes are connected by one or more directional transitions. Every state can be specified its actions, and every transition can be specified the transition condition expressions, the own actions and the priority. A local connector breaks a transition line and connected through the same name. A global connector only connects a source end of a transition called global transition. The global connector expresses briefly connection from all of the states with the same transition condition.

A structural schematic diagram (SCD) model is like the DFD model, but there are following differences: (1) Since a block expresses a circuit component, it may have a special shape (i.e., the symbols of AND, OR, DFF, ADDER, etc.), and may have some input ports and some output ports. A signal connects with special ports of the block. (2) A signal is a net, which may have several input ports and/or several output ports. (3) There is an additional graphical element -- *external-port*, so no signal has disconnection ports. A block in SCD can also be defined with all of the VDES editors, in behavioral or structural.

2) Hierarchical Design

VDES supports a hierarchical and mixed type of diagram and mixed level design. A functional module can be defined by the design of the lower level, called sub-design.

In DFD and SCD model, a block is as a functional module, which can be defined by all of DFD, CFD, FSM, TXT or other VHDL description program.

In CFD, each action is a functional module. The sub-design can only be defined by CFD.

In FSM, each state is a functional module. The sub-design can only be defined by FSM.

VDES enables mixed-level description by describing on hierarchical levels. The primary advantage of DFD, CFD and FSM lies on the ability to describe hardware on the behavior level. Furthermore, DFD and SCD are perfectly suitable for the structure descriptions, i.e., the register transfer level or the gate level. On the levels under the behavior level, a data flow represents a bus and an interconnection wire, a process or an operation represents a function unit, a gate or multiplex, and a data store represents a register.

VDES allows rebinding of a design to incorporate a different implementation of a module. A module may be described by VDES in lower hierarchical level, but it may be bound to a specific module in the library by the name of a module configuration. With the feature of hierarchical description, VDES support the development of large scale design and design reuse.

3) The Concept of *Flow*

The concept of *flow* is an abstraction on data communications between two components, states, processes, operations, or data stores. We use the concept of *flow* to build a consistent graphical model of all of the graphical editors.

At least one end of each *flow* must be connected to a component, a block or a state. The other end may be connected to a data store, a branch, another component, state, process or operation, or an external port. In the last case, the source or destination of the flow is interpreted as being external to the diagram.

C. Generating the VHDL Description

In order to make the graphical editors object-oriented, VDES first generates internal logic data which are independent to the HDL, and then translates the internal description to VHDL, or to other HDL, such as Verilog.

Before generating the internal functional data, the first step is to do the design rule checking: checking the attributes of the graphical objects; checking the connection of the flows with blocks or nodes; checking the flow network whether valid or not.

The graphical data file is dependent on the corresponding diagrams. For example, in FSM, it consists of package use information, interface information, generic parameters, local data types, local signals, variables, state machine description, and so on. Some internal graphical data formats are close to VHDL, but the state machine description is different from VHDL.

Each diagram is translated into a VHDL entity and an architecture. The interface is translated to an entity, while the types, signals, variables and the actions are translated to the corresponding elements of an architecture.

In DFD, each block (module) is translated to a component instance. Its sub-design must be an entity.

In CFD, the top design is an entity, but all of the sub-CFD are translated to several procedures. The flows in CFD are just control sequences, which need not be translated into any VHDL elements.

The actions of a state or a transition are translated to a procedure, and the state machine is translated to a process including a case structure. If it is a synchronous state machine, a process of the clock and reset signal actions must be added.

III. Graphical Object-Oriented Visual VHDL Debugger

Since a complex, hierarchical circuit design in VHDL generated from VDES diagrams is quite cumbersome to understand, a powerful debugger is imperative in our VHDL design environment. We provide a graphical object-oriented visual VHDL debugger (VDBG). In the debugger, the graphical objects in the diagram may be the debug targets. It controls the simulation. The graphical objects can be specified as observation points and interruption points to control the simulation process. The simulation result is displayed in the waveform window and back-annotated into the graphical objects.

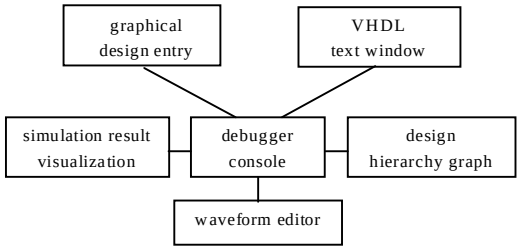

Fig 1. VDBG subsystem

A. The architecture of the VHDL debugger

The environment is chiefly constructed with a debugger console and some design entry windows. In VIDE, all design entry tools may work in two modes: editing mode and simulating mode. In the editing mode, they are used to provide to designers a means to express their design ideas, but in the simulating mode, they can interact with the debugger console to build a visual debugging environment.

The architecture of the VHDL debugging environment is shown in Figure 1.

The kernel of debugger console is a simulator, and designers can control the simulation process via it. While simulating, debugger console starts the waveform editor, and it can also start multiple design entry windows automatically and open the original design files simultaneously. The designer can operate in the debugger console window or the design entry window, such as making breakpoints, displaying signal values. The simulation process will be visual in the appreciative design entry window. All operation messages will be displayed in the console command analyzer. It guarantees to generate the same effort for same operations in different windows. During debugging process, the values of all signals in the circuit will be written to a memory file in delta format. The waveform editor can read these values from a pipe, then refreshes the signal waveform in the display window.

There are also two other important windows for debugging: data visual simulation result window and design hierarchy browser window. They can make the debugging easier.

As a hardware description language, one of the VHDL features is that it supports for design sharing and hierarchical design methodology. According to a specific configuration, a hardware may have different

constitution. For instance, in VHDL, an entity can map into two different components, and two entities can map into one component. Therefore, for a multi-level design, it becomes difficult for designers to comprehend the construction of the practical circuit, especially when the circuit scale is large. Design Hierarchy Graph (DHG) is aimed to solve the problem. All information about circuit components and their connection can be extracted from the simulation data. The debugger console can analyze it and generate a whole design hierarchy graph. The designer can browse the graph, open or close specified component. When a component is opened, its design description will be displayed.

Simulation result visualization is another means to display simulation status and data. It is a valid way to express the sophisticated data type and the relationship between objects.

B. Hierarchical Mixed-Level Simulator with Debugging

The simulator is the kernel of the debugger. In order to implement the debugging function, the simulator uses hierarchical mixed-level circuit model, which remains the component reference structure and mixed level components. The advantage of the model is that it reduces the space occupied, and it is easy to get all VHDL source information during debugging operations.

In a circuit model, there is a top module and some sub-modules referenced hierarchically. The top module or each sub-module is an entity with a corresponding architecture. Each architecture may be described in the behavioral or structural level, and may be in the system level, the register transfer level, the gate level or the switch level. The simulator runs the top module and then calls the sub-modules when executing the component instance statement.

Because the hierarchical model is a hierarchical network but not a real tree, a sub-module may have several parent modules, so it has several calling paths. Therefore, when a sub-module is executed, the values of the local signals and the variables are different depending on the current calling path. In addition, when the processes in the VHDL description are reactivated or the simulator continues after interruption, the execution must recognize the current point with the calling path, and be continued.

The simulator is isolated with entry tools. It is only able to recognize the VHDL description. A break point in the VHDL description may be at one of the following positions: at a simulation time; on a statement line; in an architecture, a block, a process, a procedure or a function; on a signal or variable; when a condition is true. Besides, it may be traced on each step and displayed the state messages.

C. Setting Break Point In Design Entry Window

Although the simulator orients to the VHDL description file, VIDE supports a graphical object-oriented debugging approach. The designer can set break points in the diagrams of the design entry window. After simulation, the simulation state can be back-annotated in the corresponding diagram.

When the simulation is running, the corresponding editor window goes to simulating mode. In this mode, can set break points and send them to the simulator.

Every graphical object can be as a break point. VDES transforms the graphical break point to a textual breakpoint in VHDL program. For example, a graphical break point in a block of the DFD can be transformed to a textual break point at the component instanciation statement line, and a break point at a state of the FSM is transformed to a statement line in a case structure.

VDES sends the transformed textual point to VDBG. VDBG manages all of the textual break points, and control the simulation.

When the simulation is interrupted caused by a break point, VDBG gets the simulation status and sends it to VDES. VDES transforms the information and displays the status in the diagram.

The VHDL file generated from VDES or entered into text editor can be open in debug console window. The designer can also set break point in debug window. The simulation dynamic status shows in the window simultaneously.

IV. CONCLUSION

In this paper, we showed a graphical object-oriented design entry and debugging approach. Using VIDE, designer can enter his design in diagram approach. While simulation is running, he can control and observe the simulation status in the design entry window or in the textual debug window, so it is easy to verify his design and correct the mistakes.

REFERENCES

[1] Reda A. Ammar and PE Roaiene. “Visualizing a Hierarchy of Performance Model for Software Systems”,*Software Practice and Experience*, March 1993.

[2] Winchester and Sally Jane, "Modeling Language To Become An Expert Logic Circuit Debugger", *DAI-B*, Jun 1994.

[3] Bergman and Lawreace D., "VIEW: A System For Prototyping Scientific Visualizations", *DAI-B*, Oct. 1993.

[4] D.S.Boning, M.L.Heytens, and A.S. Wong, “The Intertool Profile Interchange Format: An Object-Oriented Approach”, *IEEE Transactions on CAD of IC&S*, Sept 1991.

[5] Take Shimonura and Saduhiro Isoda, “linked-list Visualization for Debugging”, *IEEE/SOFTWARE*, May 1991.