WL-TR-91-5018

A PROLOG SYSTEM FOR CONVERTING VHDL-BASED MODELS TO GENERALIZED EXTRACTION SYSTEM (GES) RULES

Michael Alan Dukes, M.S.E.E.
Captain, U.S. Army
Air Force Institute of Technology

Frank Markham Brown, PhD
Professor of Electrical Engineering
Air Force Institute of Technology

Joanne E. Degroat, PhD
Assistant Professor of Electrical Engineering
Ohio State University

Design Branch
Microelectronics Division

June 1991

Final Report For Period June 1990 TO December 1990

SOLID STATE ELECTRONICS DIRECTORATE
WRIGHT LABORATORY
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6543

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

MICHAEL A. DUKES, Capt, USA
Air Force Institute of Technology

JOHN W. HINES, Chief
Design Branch
Microelectronics Division

FOR THE COMMANDER

STANLEY E. WAGNER, Chief
Microelectronics Division
Electronic Technology Laboratory

If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify WL/ELED, WPAFB, OH 45433-6543 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Jun 91	FINAL Jun 90-Dec 90

4. TITLE AND SUBTITLE
A Prolog System for Converting VHDL-Based Models to Generalized Extraction System (GES) Rules

5. FUNDING NUMBERS
PE 62204F
PR 6096
TA 40
WU 18

6. AUTHOR(S)
Michael Alan Dukes, Frank Markham Brown, Joanne E. DeGroat

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AFIT/ENG
Wright-Patterson AFB OH 45433-6543

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
S.S. Electronics Directorate
WL, AFSC
WL/ELED
Wright-Patterson AFB OH 45433-6543
Michael Dukes, 255-8629

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
WL-TR-91-5018

11. SUPPLEMENTARY NOTES
The computer software contained herein are "harmless." Already in the public domain.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
A Prolog System, called vhdl2ges, is presented. vhdl2ges translates structural VHDL models to GES extraction rules. Acceptable VHDL modes for vhdl2ges are described. Enhancements to GES extraction rules are also covered. Finally, limitations of hierarchical extraction are considered.

14. SUBJECT TERMS
Formal Hardware Verification - Logic Programming - VLSI - VHDL

15. NUMBER OF PAGES
44

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

Table of Contents

List of Figures

I. Introduction

With the advent of VHDL [1], accurate documentation of hardware designs is a practical reality. In the past, schematics were typically used for documentation of hardware designs. However, these schematics would usually become obsolete as the hardware design was being constructed. Deviations in the hardware design would sometimes not be reflected in the schematics. For design groups, failure to update schematics could lead to different parts of a hardware design becoming incompatible.

A system to ensure compliance of hardware with its VHDL documentation is presented in this paper. The system, ***vhdl2ges***, is meant to help guide the development of hardware by pointing out where hardware deviates from its VHDL documentation. A tool called *vhdl_parser*©[1] [2] is used to parse the VHDL description into a Prolog-type intermediate form *vhdl2ges* translates the structural VHDL description from this Prolog-type intermediate form to extraction rules for a system called a generalized extraction system (GES) [3]. A "customized" GES is constructed that checks a netlist derived from a layout description for deviations from the components and interconnections specified in the structural VHDL documentation. The end result is an accurate VHDL document representing the constructed hardware. Furthermore, *vhdl2ges* makes the VHDL documentation the driving element in the hardware development.

The form of extraction performed by GES entails identifying higher level components constructed from existing lower level components. Extraction in GES is a three-step process performed iteratively on a component netlist. The first step is to find a group of related components based on the component types and interconnections between them. Next, the identified components are eliminated from the component netlist. Finally, a new higher-level component (constructed from the identified components) is added to the component netlist. In its simplest form, the extraction process may be viewed as identifying digital logic gates formed from a transistor netlist. The next level might involve identification of half adders from a component netlist of digital logic gates. The extraction process in GES may be started at any level to produce a higher and more abstract level of representation.

Both *vhdl2ges* and GES have been tested using Quintus Prolog[2] under Ultrix[3]. Every attempt has been made to adhere to Prolog standards specified by Clocksin and Mellish [4]. GES was originally developed to extract VLSI custom layout designs generated in *magic* using the Berkeley Distribution of Design Tools [5]. Transistor netlists were generated from *magic* using either *extract* or *cif*[4]. Netlists from other computer aided design (CAD) tools may be input into GES after processing by an input filter which produces the netlist format required by GES.

The purpose of this report is to describe the acceptable VHDL code for *vhdl2ges*, to describe the Prolog routines in *vhdl2ges*, and to describe the changes made to GES. The type of structural VHDL models that are accepted may have any number of hierarchical levels. Enhancements to GES are discussed in Section 2. The style of structural VHDL that is used by *vhdl2ges* is described in Section 3. Some examples of how structural VHDL maps to GES are also described in Section 3. Section 4 is a description of the Prolog program that accepts the VHDL intermediate form from *vhdl_parser*.

[1] Copyright 1990 by the Microelectronics Center of North Carolina

[2] Quintus and Quintus Prolog are trademarks of Quintus Computer Systems, Inc.

[3] Ultrix is a trademark of Digital Equipment Corporation

[4] CALTECH Intermediate Format (CIF)

II. Changes to GES and VHDL_PARSER

2.1 Changes to GES

This section describes the changes made to GES. These changes were necessary to help make extraction rule generation easier and to correct shortfalls of the original GES system described in [3].

Previously, the format for a GES [3] extraction rule was reported as the following.

head :-
  matching_goal$_1$,
    .
    .
    .
  matching_goal$_n$,
  not_connected([*internal_argument_list*]),
  retract_goal$_1$,
    .
    .
    .
  retract_goal$_n$,
  asserta(*head*(*argument_list*)),
  fail.
head.

The following is the format for the new extraction rule.

head :-
  matching_goal$_1$,
    .
    .
  matching_goal$_n$,
  unique_component([*XYs*]),
  not_connected([*internal_argument_list*],
    external_argument_list]),
  retract_goal$_1$,
    .
    .
  retract_goal$_n$,
  asserta(*head*(*argument_list*)),
  fail.
head.

The extraction rule is based upon six procedural steps:

1. Identify the component from its lower-level components.
2. Ensure that the identified lower-level components are unique.
3. Check that the values of internal nodes do not match other nodes.
4. Delete the lower-level components from the component netlist.
5. Add the newly found component to the component netlist.
6. Check to see if there are more lower-level components.

Step 1 must occur first, step two must occur second, step three must occur third, and step 6 must occur last; however, the order of steps 4 and 5 is not important. Within the extraction rule are two Prolog routines that require further explanation.

The rule **unique_component/1** is true *iff* all of the components corresponding to all of the goals, $matching\_goal_1$ through $matching\_goal_n$, have unique (X,Y) locations. In Prolog, **unique_component/1** is expressed as the following.

```
unique_component([]) :- !.
unique_component([XY|XYs]) :-
  not_member(XY,XYs),unique_component(XYs).

not_member(_,[]) :- !.
not_member(Node,[Head|Tail]) :-
  Node \== Head,not_member(Node,Tail).
```

The Prolog rule **not_member/2** is used to ensure that no two (X,Y) coordinate locations are equal.

The Prolog rule **not_connected/2** was modified from the original GES. The following is the new **not_connected/2**.

```
not_connected([],_) :- !.
not_connected([Node|Tail],External) :-
  not_member(Node,Tail),not_member(Node,External),
  not_connected(Tail,External).
```

The rule **not_connected/2** is true *iff* none of the internal nodes are interconnected and none of the internal nodes are connected to external nodes. The rule **not_connected/2** calls upon the same **not_member/2** Prolog rule as **unique_component/1**.

2.2 Changes to VHDL_PARSER

There were two changes made to *vhdl_parser*. The changes concerned VHDL file naming conventions and case sensitivity of the letters used in a VHDL description.

The Prolog routine **file_path/2** was originally the following.

```
file_path(Name,File) :- concatenate(['data/',Name,'.vhdl'],File).
file_path(Name,File) :- concatenate([Name,'.vhdl'],File)
```

file_path/2 was changed to the following.

```
file_path(Name,File) :- concatenate(['data/',Name,'.vhdl'],File).
file_path(Name,File) :- concatenate([Name,'.vhdl'],File).
file_path(Name,File) :- concatenate([Name,'.vhd'],File).
```

The change allowed for VHDL file extensions of **.vhd** as well as **.vhdl**.

The second change made to *vhdl_parser* causes all upper and lower case letters read in to be forced to lower case. This change was necessary since VHDL is case insensitive and Prolog is case sensitive. The Prolog routines **getmy0/1** and **getmy/1** were substituted for get0/1 and get/1. The Prolog code for the new routines follows.

```
getmy0(Char) :-
  get0(CharUp),
  to_lower(CharUp,Char).

getmy(Char) :-
  get(CharUp),
  to_lower(CharUp,Char).
```

Both Prolog routines call on a Quintus Prolog library routine called to_lower/2 [6].

III. Converting VHDL Descriptions to GES

This section describes the acceptable VHDL syntax for a VHDL description and the form of the GES code produced. Since the purpose of *vhdl2ges* is to search for components and interconnections, VHDL-constructs that do not provide information to assist in this search are ignored.

3.1 Acceptable VHDL Syntax

This section details the VHDL language constructs accepted by *vhdl2ges*. Several examples of acceptable structural VHDL models are provided. The VHDL language constructs are taken from the Syntax Summary of [2]. At a minimum, the VHDL description must contain the following.

entity identifier **is**
  *formal*_port_clause
  end *entity*_simple_name ;

architecture identifier **of** *entity*_name **is**
  component identifier
    *local*_port_clause
    end component;
  begin
  *instantiation*_label :
    *component*_name port_map_aspect ;
  end *architecture*_simple_name ;

Below is an example of a VHDL description conforming to the above description.

```
entity comp is
  port (A : in bit);
  end comp;

architecture structure of comp is
  component sub_comp
    port (A : in bit);
    end component;
  begin
  sub_comp00 : sub_comp port map (A);
  end structure;
```

Additional VHDL language constructs supported are shown below.

signal identifier_list : subtype_indication;

alias identifier : subtype_indication **is** name;

All other VHDL language constructs are ignored.

3.2 *Examples of Acceptable VHDL Descriptions*

Some further examples of acceptable structural VHDL models are shown below.

```
entity stage2 is
  port (
      X_vector : in bit_vector(3 downto 0);
      S_vector : in bit_vector(2 downto 0);
      Y        : in bit;
      Result   : out bit_vector(4 downto 0)
     );

  end stage2;

architecture stage2 of stage2 is

         component stage1
              port (
                  X_vector : in bit_vector(3 downto 0);
                  Y        : in bit;
                  Result   : out bit_vector(3 downto 0)
                 );
            end component;

         component full_adder
          port (
              X, Y, Cin: in bit;
              Sum, Cout: out bit
             );
            end component;

         component half_adder
          port (
              X, Y     : in bit;
              Sum, Cbar: out bit
             );
            end component;

       alias S_0         : bit is S_vector(0);
       alias S_1         : bit is S_vector(1);
       alias S_2         : bit is S_vector(2);
       alias R_0         : bit is Result(0);
       alias R_1         : bit is Result(1);
       alias R_2         : bit is Result(2);
       alias R_3         : bit is Result(3);
       alias R_4         : bit is Result(4);

       signal A          : bit_vector (3 downto 0);
       alias A_0         : bit is A(0);
       alias A_1         : bit is A(1);
       alias A_2         : bit is A(2);
```

```
      alias A_3         : bit is A(3);
      signal HA1        : bit;
      signal C_2,C_3    : bit;

      begin
         C0: STAGE1 port map (X_vector, Y, A);

         C00: half_adder port map (S_0,A_0,R_0,HA1);
         C01: full_adder port map (S_1,A_1,HA1,R_1,C_2);
         C02: full_adder port map (S_2,A_2,C_2,R_2,C_3);
         C03: half_adder port map (C_3,A_3,R_3,R_4);
    end stage2;



entity mult is
  port (
        X_vector : in bit_vector(3 downto 0);
        Y_vector : in bit_vector(3 downto 0);
        Result   : out bit_vector(7 downto 0));
  end mult;

architecture mult of mult is

         component stage1
              port (
                  X_vector : in bit_vector(3 downto 0);
                  Y        : in bit;
                  Result   : out bit_vector(3 downto 0)
                 );
            end component;

         component stage2
              port (
                  X_vector : in bit_vector(3 downto 0);
                  S_vector : in bit_vector(2 downto 0);
                  Y        : in bit;
                  Result   : out bit_vector(4 downto 0)
                 );
            end component;

         component stage3
              port (
                  X_vector : in bit_vector(3 downto 0);
                  S_vector : in bit_vector(3 downto 0);
                  Y        : in bit;
                  Result   : out bit_vector(4 downto 0)
                 );
            end component;

        signal S1_2       : bit_vector(2 downto 0);
```

```
    signal S2_3       : bit_vector(3 downto 0);
    signal S3_4       : bit_vector(3 downto 0);

begin

  C0: stage1 port map(X_vector => X_vector,
                      Y => Y_vector(0),
                      Result(3 downto 1) => S1_2,
                      Result(0) => Result(0));

  C1: stage2 port map(X_vector => X_vector,
                      S_vector => S1_2,
                      Y => Y_vector(1),
                      Result(4 downto 1) => S2_3,
                      Result(0) => Result(1));

  C2: stage3 port map(X_vector => X_vector,
                      S_vector => S2_3,
                      Y => Y_vector(2),
                      Result(4 downto 1) => S3_4,
                      Result(0) => Result(2));

  C3: stage3 port map(X_vector => X_vector,
                      S_vector => S3_4,
                      Y => Y_vector(3),
                      Result => Result(7 downto 3));

end mult;
```

Only the VHDL information pertaining to components and interconnections is considered. As ***vhdl2ges*** is presently constructed, only one entity/architecture pair for a structural VHDL model is allowed within a file.

IV. The VHDL2GES System

4.1 The VHDL2GES Prolog Program

The following is an explanation of the Prolog routines used to translate the VHDL intermediate form created by *vhdl_parser* into extraction rules used by GES. Some familiarity with Prolog is assumed. *vhdl2ges* works strictly with the intermediate form, therefore an assumption in using this tool is that the VHDL description to be converted has already been parsed by *vhdl_parser*.

The Prolog routine **vhdl2ges/1** is the main driver of *vhdl2ges*.

```
vhdl2ges(File) :-
  vhdl_read(File),
  vhdl2ges_sub,
  halt.
```

The Prolog routine **vhdl_read/1** is part of the *vhdl_parser* system. The name of the VHDL file to be parsed is passed to **vhdl_read/1**. Afterwards, **vhdl2ges_sub/0** is called to work on the intermediate form generated by *vhdl_parser*.

vhdl2ges_sub/0 calls a series of Prolog routines.

```
vhdl2ges_sub :-
  vhdl2ges_comp_ent,
  vhdl2ges_comp_arch,
  vhdl2ges_gen_rule,
  vhdl2ges_gen_find_anomaly,
  vhdl2ges_gen_rule_name,
  vhdl2ges_gen_listing,
  nl.
```

Some utilities common to Prolog routines in *vhdl2ges* are shown below.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     vhdl2ges_assert_comp_vector_list(p1,p2,p3,p4,p5).
%     This routine converts a list of bit_vector names (implied equal ranges),
%     to a list of element names.  A table is built in memory called
%     vhdl2ges_comp_vector_table.  Each record of the table contains three
%     fields denoting the name of the component, the name of the vector,
%     and a list of its expanded elements.
%
%     Parameters:
%
%         p1:   component name
%         p2:   list of bit_vector names
%         p3:   high index
%         p4:   low index
%         p5:   returned element list.
%
%     Calls:
%         append/3,vhdl2ges_expand_name/4,assert/1.
%
%     Example:
%
%         | ?- vhdl2ges_assert_comp_vector_list(stage1,[a,b,c],8,4,X).
%
%         X = [a_8,a_7,a_6,a_5,a_4,b_8,b_7,b_6,b_5,b_4,c_8,c_7,c_6,c_5,c_4] ;
%
%         no
%         | ?- listing(vhdl2ges_comp_vector_table).
%
%         vhdl2ges_comp_vector_table(stage1,a,[a_8,a_7,a_6,a_5,a_4]).
%         vhdl2ges_comp_vector_table(stage1,b,[b_8,b_7,b_6,b_5,b_4]).
%         vhdl2ges_comp_vector_table(stage1,c,[c_8,c_7,c_6,c_5,c_4]).

vhdl2ges_assert_comp_vector_list(_,[],_,_,[]) :- !.
vhdl2ges_assert_comp_vector_list(Comp,[Head|VarList],High,Low,VarIntList) :-
  !,
  vhdl2ges_expand_name(Head,High,Low,NewList),
  assert(vhdl2ges_comp_vector_table(Comp,Head,NewList)),
  vhdl2ges_assert_comp_vector_list(Comp,VarList,High,Low,IntList),
  append(NewList,IntList,VarIntList).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    vhdl2ges_assert_vector_list(p1,p2,p3,p4).
%    This routine converts a list of bit_vector names (implied equal ranges),
%    to a list of element names.  A table is built in memory called
%    vhdl2ges_vector_table.  Each record of the table contains two
%    fields denoting the name of the vector and a list of its expanded
%    elements.
%
%    Parameters:
%
%        p1:   list of bit_vector names
%        p2:   high index
%        p3:   low index
%        p4:   returned element list.
%
%    Calls:
%        append/3,vhdl2ges_expand_name/4,assert/1.
%
%    Example:
%
%        | ?- vhdl2ges_assert_vector_list([a,b],22,20,X).
%
%        X = [a_22,a_21,a_20,b_22,b_21,b_20] ;
%
%        no
%        | ?- listing(vhdl2ges_vector_table).
%
%        vhdl2ges_vector_table(a,[a_22,a_21,a_20]).
%        vhdl2ges_vector_table(b,[b_22,b_21,b_20]).

vhdl2ges_assert_vector_list([],_,_,[]) :- !.
vhdl2ges_assert_vector_list([Head|VarList],High,Low,VarIntList) :-
  !,
  vhdl2ges_expand_name(Head,High,Low,NewList),
  assert(vhdl2ges_vector_table(Head,NewList)),
  vhdl2ges_assert_vector_list(VarList,High,Low,IntList),
  append(NewList,IntList,VarIntList).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    vhdl2ges_expand_name_list(p1,p2,p3,p4).
%    This routine converts a list of bit_vector names (implied equal ranges),
%    to a list of element names.
%
%    Parameters:
%
%        p1:   list of bit_vector names
%        p2:   high index
%        p3:   low index
%        p4:   returned element list.
%
%    Calls:
%        append/3,vhdl2ges_expand_name/4.
%
%    Examples:
%
%        | ?- vhdl2ges_expand_name_list([a_in,b_in],1050,1048,X).
%
%        X = [a_in_1050,a_in_1049,a_in_1048,b_in_1050,b_in_1049,b_in_1048] ;
%
%        no
%

vhdl2ges_expand_name_list([],_,_,[]) :- !.
vhdl2ges_expand_name_list([Head|VarList],High,Low,VarIntList) :-
  !,
  vhdl2ges_expand_name(Head,High,Low,NewList),
  vhdl2ges_expand_name_list(VarList,High,Low,IntList),
  append(NewList,IntList,VarIntList).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    vhdl2ges_expand_name(p1,p2,p3,p4).
%    This routine converts a bit_vector name to a list of element names.
%
%    Parameters:
%
%        p1:   bit_vector name
%        p2:   high index
%        p3:   low index
%        p4:   returned element list.
%
%    Calls:
%        append/3,name/2
%
%    Examples:
%
%        | ?- vhdl2ges_expand_name(a_in,100,95,X).
%
%        X = [a_in_100,a_in_99,a_in_98,a_in_97,a_in_96,a_in_95] ;
%
%        no
%        | ?- vhdl2ges_expand_name(a_in,95,100,X).
%
%        no
%
%     In the second example, the range was ascending rather than
%     descending.

vhdl2ges_expand_name(Var,Low,Low,[NewList]) :-
  !,
  name(Low,LowList),
  name(Var,VarList),
  append([95],LowList,Tail),
  append(VarList,Tail,NewVarList),
  name(NewList,NewVarList).
vhdl2ges_expand_name(Var,High,Low,[NewASCList|NewList]) :-
  High > Low,
  name(High,HighList),
  name(Var,VarList),
  append([95],HighList,Tail),
  append(VarList,Tail,NewVarList),
  name(NewASCList,NewVarList),
  Next is High - 1,
  vhdl2ges_expand_name(Var,Next,Low,NewList).
```

The following Prolog routine, **vhdl2ges_comp_ent/0**, is called by **vhdl2ges_sub**.

```
vhdl2ges_comp_ent :-
  vhdl2ges_find_entity(Identifier,Interface),
  vhdl2ges_interface_list(Interface,SigList),
  assert(component(Identifier,SigList)),
  assert(external_sig(SigList)).
```

From the intermediate form generated by *vhdl_parser* it constructs two facts to be asserted on the Prolog facts database. The first one, **component/2**, has the following format.

component(*entity_name*,[*port_list*]).

An example of how **component/2** might look would be the following.

component(half_adder,[x,y,sum,cbar]).

Another fact asserted as a result of calling **vhdl2ges_comp_ent/0** is **external_sig/1**. Its format is the following.

external_sig([*port_list*]).

The Prolog routine **vhdl2ges_find_entity/2** is used by **vhdl2ges_comp_ent/0** to return the entity name and port list of the component being converted to an extraction rule.

```
vhdl2ges_find_entity(Identifier,Interface) :-
  design_unit(_,entity(Identifier,_,Interface,[],[])).
```

The Prolog routine **vhdl2ges_interface_list/2** is called by **vhdl2ges_comp_ent/0** to expand port names from their bit-vector constructions into single-element name-constructions. A table is built called **vhdl2ges_vector_table/2**.

```
vhdl2ges_interface_list([],[]) :- !.
vhdl2ges_interface_list(
    [interface_element(_,VarList,_,
      vhdl_subtype(_,_,index([vhdl_range(Low,to,High)])),null,null)
      |IntList],
    NewList) :-
  !,
  vhdl2ges_assert_vector_list(VarList,High,Low,VarIntList),
  vhdl2ges_interface_list(IntList,RetList),
  append(VarIntList,RetList,NewList).
vhdl2ges_interface_list(
    [interface_element(_,VarList,_,
      vhdl_subtype(_,_,index([vhdl_range(High,downto,Low)])),null,null)
      |IntList],
    NewList) :-
  !,
  vhdl2ges_assert_vector_list(VarList,High,Low,VarIntList),
  vhdl2ges_interface_list(IntList,RetList),
  append(VarIntList,RetList,NewList).
vhdl2ges_interface_list(
    [interface_element(_,VarList,_,_,null,null)|IntList],
    NewList) :-
  !,
  vhdl2ges_interface_list(IntList,RetList),
  append(VarList,RetList,NewList).
vhdl2ges_interface_list([_|IntList],RetList) :-
  !,vhdl2ges_interface_list(IntList,RetList).
```

vhdl2ges_comp_arch/0 is used to convert language constructs in the VHDL architecture to facts in the Prolog database for further processing.

```
vhdl2ges_comp_arch :-
  component(EntName,_),
  vhdl2ges_find_arch(EntName,Decl,Body),
  vhdl2ges_create_int_sig_list(Decl,IntList),
  assert(internal_sig(IntList)),
  vhdl2ges_create_alias_table(Decl),
  vhdl2ges_create_comps(Decl),
  vhdl2ges_create_goals(Body).
```

The Prolog routine **vhdl2ges_find_arch/3** returns the list of architecture declarative elements and architectural body elements for use in **vhdl2ges_comp_arch/0**. The entity name passed to **vhdl2ges_find_arch/3** is used to find the correct architecture.

```
vhdl2ges_find_arch(EntName,Decl,Body) :-
  design_unit(_,arch(_,EntName,Decl,Body)).
```

The Prolog routine **vhdl2ges_create_int_sig_list/2** is called by **vhdl2ges_comp_arch/0**. The purpose of **vhdl2ges_create_int_sig_list/2** is to create a signal list from the signals declared in the VHDL architecture declarative region. **vhdl2ges_assert_vector_list/4** is a *vhdl2ges* utility called by **vhdl2ges_create_int_sig_list/2**.

```
vhdl2ges_create_int_sig_list([],[]) :- !.
vhdl2ges_create_int_sig_list(
    [object_declaration(signal,SigList,
        vhdl_subtype(null,_,index([vhdl_range(High,downto,Low)])),null,null)|
    DeclList],IntList) :-
  !,
  vhdl2ges_assert_vector_list(SigList,High,Low,IntermList),
  vhdl2ges_create_int_sig_list(DeclList,RetList),
  append(IntermList,RetList,IntList).
vhdl2ges_create_int_sig_list(
    [object_declaration(signal,SigList,
        vhdl_subtype(null,_,index([vhdl_range(Low,to,High)])),null,null)|
    DeclList],IntList) :-
  !,
  vhdl2ges_assert_vector_list(SigList,High,Low,IntermList),
  vhdl2ges_create_int_sig_list(DeclList,RetList),
  append(IntermList,RetList,IntList).
vhdl2ges_create_int_sig_list([object_declaration(signal,SigList,_,null,null)|
    DeclList],IntList) :-
  !,
  vhdl2ges_create_int_sig_list(DeclList,RetList),
  append(SigList,RetList,IntList).
vhdl2ges_create_int_sig_list([_|DeclList],IntList) :-
  !,vhdl2ges_create_int_sig_list(DeclList,IntList).
```

The Prolog routine **vhdl2ges_create_alias_table/1** is called by **vhdl2ges_comp_arch/0**. It is used to create alias tables from the aliases declared in the VHDL architecture declarative region. The alias tables are used later to translate alias names into their actual names. A number of helper routines are called to accomplish this objective. The routines are **vhdl2ges_assert_vector_list/4**, **vhdl2ges_expand_name_list/4**, and **vhdl2ges_assert_alias_table/2**. Both Prolog routines **vhdl2ges_assert_vector_list/4** and **vhdl2ges_expand_name_list/4** are utilities described previously.

```
vhdl2ges_create_alias_table([]) :- !.
vhdl2ges_create_alias_table([vhdl_alias(Alias,
    vhdl_subtype(_,_,index([vhdl_range(AliasHigh,downto,AliasLow)])),
    vhdl_name(prefix(Name),vhdl_range(NameHigh,downto,NameLow)) )
    |Decl]) :-
  !,
  vhdl2ges_assert_vector_list([Alias],AliasHigh,AliasLow,AliasList),
  vhdl2ges_expand_name_list([Name],NameHigh,NameLow,NameList),
  vhdl2ges_assert_alias_table(AliasList,NameList),
  vhdl2ges_create_alias_table(Decl).
vhdl2ges_create_alias_table([vhdl_alias(Alias,
    vhdl_subtype(_,_,index([vhdl_range(AliasLow,to,AliasHigh)])),
    vhdl_name(prefix(Name),vhdl_range(NameLow,to,NameHigh)) )
    |Decl]) :-
  !,
  vhdl2ges_assert_vector_list([Alias],AliasHigh,AliasLow,AliasList),
  vhdl2ges_expand_name_list([Name],NameHigh,NameLow,NameList),
  vhdl2ges_assert_alias_table(AliasList,NameList),
  vhdl2ges_create_alias_table(Decl).
vhdl2ges_create_alias_table([vhdl_alias(Alias,
    vhdl_subtype(_,_,index([vhdl_range(AliasHigh,downto,AliasLow)])),
    Name )
    |Decl]) :-
  !,
  vhdl2ges_assert_vector_list([Alias],AliasHigh,AliasLow,AliasList),
  vhdl2ges_vector_table(Name,NameList),
  vhdl2ges_assert_alias_table(AliasList,NameList),
  vhdl2ges_create_alias_table(Decl).
vhdl2ges_create_alias_table([vhdl_alias(Alias,
    vhdl_subtype(_,_,index([vhdl_range(AliasLow,to,AliasHigh)])),
    Name )
    |Decl]) :-
  !,
  vhdl2ges_assert_vector_list([Alias],AliasHigh,AliasLow,AliasList),
  vhdl2ges_vector_table(Name,NameList),
  vhdl2ges_assert_alias_table(AliasList,NameList),
  vhdl2ges_create_alias_table(Decl).
vhdl2ges_create_alias_table([vhdl_alias(Alias,_,
    vhdl_name(prefix(Name),[Number],_) )|Decl]) :-
  !,
  vhdl2ges_expand_name_list([Name],Number,Number,NameList),
  vhdl2ges_assert_alias_table([Alias],NameList),
  vhdl2ges_create_alias_table(Decl).
```

```
vhdl2ges_create_alias_table([vhdl_alias(Alias,_,Sig)|Decl]) :-
  !,
  vhdl2ges_assert_alias_table([Alias],[Sig]),
  vhdl2ges_create_alias_table(Decl).
vhdl2ges_create_alias_table([_|Decl]) :-
  !,
  vhdl2ges_create_alias_table(Decl).
```

vhdl2ges_assert_alias_table/2 is called by **vhdl2ges_create_alias_table/1** for the purpose of creating a lookup table of aliases.

```
vhdl2ges_assert_alias_table([],[]) :- !.
vhdl2ges_assert_alias_table(
    [AliasHead|AliasTail],[ActualHead|ActualTail]) :-
  !,
  assert(vhdl2ges_alias_table(AliasHead,ActualHead)),
  vhdl2ges_assert_alias_table(AliasTail,ActualTail).
```

An example of how **vhdl2ges_assert_alias_table/2** works is the following.

```
| ?- vhdl2ges_assert_alias_table([a_1,a_0],[s_20,s_19]).

yes
| ?- listing(vhdl2ges_alias_table).

vhdl2ges_alias_table(a_1,s_20).
vhdl2ges_alias_table(a_0,s_19).

yes
| ?-
```

The information conveyed by the table above is that **a_1** is an alias for **s_20**.

The Prolog routine **vhdl2ges_create_comps/1** is called by **vhdl2ges_comp_arch/0** for the purpose of creating a table of component declarations from within the architecture declarative region. The Prolog routine **vhdl2ges_comp_interface_list/3** is called to generate a signal list from the component interface list in intermediate form.

```
vhdl2ges_create_comps([]) :- !.
vhdl2ges_create_comps([vhdl_comp(Comp,_,Interface)|Decl]) :-
  !,
  vhdl2ges_comp_interface_list(Comp,Interface,SigList),
  assert(comp(Comp,SigList)),
  vhdl2ges_create_comps(Decl).
vhdl2ges_create_comps([_|Decl]) :-
  !,
  vhdl2ges_create_comps(Decl).
```

The following is an example of how the component table is created from a component declaration in intermediate form.

```
| ?- vhdl2ges_create_comps([vhdl_comp(and_gate,null,[interface_element(
     signal,[x,y],in,vhdl_subtype(null,bit,null),null,null),
     interface_element(signal,[output],out,vhdl_subtype(null,bit,null),
     null,null)])]).

yes
| ?- listing(comp).

comp(and_gate,[x,y,output]).

yes
| ?-
```

vhdl2ges_comp_interface_list/3 is called by **vhdl2ges_create_comps/1** for translating the intermediate form of the interface list into a signal list.

```
vhdl2ges_comp_interface_list(_,[],[]) :- !.
vhdl2ges_comp_interface_list(Comp,
    [interface_element(_,VarList,_,
      vhdl_subtype(_,_,index([vhdl_range(Low,to,High)])),null,null)
      |IntList],
    NewList) :-
  !,
  vhdl2ges_assert_comp_vector_list(Comp,VarList,High,Low,VarIntList),
  vhdl2ges_comp_interface_list(Comp,IntList,RetList),
  append(VarIntList,RetList,NewList).
vhdl2ges_comp_interface_list(Comp,
    [interface_element(_,VarList,_,
      vhdl_subtype(_,_,index([vhdl_range(High,downto,Low)])),null,null)
      |IntList],
    NewList) :-
  !,
  vhdl2ges_assert_comp_vector_list(Comp,VarList,High,Low,VarIntList),
  vhdl2ges_comp_interface_list(Comp,IntList,RetList),
  append(VarIntList,RetList,NewList).
vhdl2ges_comp_interface_list(Comp,
    [interface_element(_,VarList,_,_,null,null)|IntList],
    NewList) :-
  !,
  vhdl2ges_comp_interface_list(Comp,IntList,RetList),
  append(VarList,RetList,NewList).
vhdl2ges_comp_interface_list(Comp,[_|IntList],RetList) :-
  !,vhdl2ges_comp_interface_list(Comp,IntList,RetList).
```

The Prolog routine **vhdl2ges_create_goals/1** is called by **vhdl2ges_comp_arch/0** for the purpose of creating in the database a table of instantiated components, "goals", for the instantiated components in the architecture body.

```
vhdl2ges_create_goals([]) :- !.
vhdl2ges_create_goals([comp_instant(_,Name,_,ElementList)|Body]) :-
  vhdl2ges_create_actual_list(Name,ElementList,ActualList),
  assert(goal(Name,ActualList)),
  !,
  vhdl2ges_create_goals(Body).
vhdl2ges_create_goals([_|Body]) :-
  !,
  vhdl2ges_create_goals(Body).
```

vhdl2ges_create_actual_list/3 is called by **vhdl2ges_create_goals/1**. The Prolog routine **vhdl2ges_create_actual_list/3** forms the interface list in the correct order for the goal table regardless of the form of the *port_map_aspect*.

```
vhdl2ges_create_actual_list(_,[],[]) :- !.
vhdl2ges_create_actual_list(CompName,
    [element(null,vhdl_name(prefix(Name),
        vhdl_range(Low,to,High)))|ElementList],ActualList) :-
  !,
  vhdl2ges_expand_name(Name,High,Low,VectorList),
  vhdl2ges_check_alias(VectorList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,IntermList),
  append(RealList,IntermList,ActualList).
vhdl2ges_create_actual_list(CompName,
    [element(null,vhdl_name(prefix(Name),
        vhdl_range(High,downto,Low)))|ElementList],ActualList) :-
  !,
  vhdl2ges_expand_name(Name,High,Low,VectorList),
  vhdl2ges_check_alias(VectorList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,IntermList),
  append(RealList,IntermList,ActualList).
vhdl2ges_create_actual_list(CompName,
  [element(null,
          vhdl_call(Name,[element(null,NameNumb)]))
    |ElementList],ActualList) :-
  !,
  vhdl2ges_expand_name(Name,NameNumb,NameNumb,NameList),
  vhdl2ges_check_alias(NameList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,InterList),
  append(RealList,InterList,ActualList).
vhdl2ges_create_actual_list(CompName,
    [element(null,Name)|ElementList],ActualList) :-
  vhdl2ges_vector_table(Name,VectorList),
  !,
  vhdl2ges_check_alias(VectorList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,IntermList),
  append(RealList,IntermList,ActualList).
```

```
vhdl2ges_create_actual_list(CompName,
    [element(null,Name)|ElementList],ActualList) :-
  !,
  vhdl2ges_check_alias([Name],Real),
  vhdl2ges_create_actual_list(CompName,ElementList,IntermList),
  append(Real,IntermList,ActualList).
vhdl2ges_create_actual_list(CompName,
  [element(vhdl_name(prefix(Assoc),vhdl_range(AssocHigh,downto,AssocLow)),
           vhdl_name(prefix(Name),vhdl_range(NameHigh,downto,NameLow)))
     |ElementList],ActualList) :-
  !,
  vhdl2ges_expand_name(Assoc,AssocHigh,AssocLow,AssocList),
  vhdl2ges_expand_name(Name,NameHigh,NameLow,NameList),
  vhdl2ges_check_alias(NameList,RealList),
  vhdl2ges_assert_element(AssocList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,InterList),
  comp(CompName,SignalList),
  vhdl2ges_retract_element(SignalList,OrderedList),
  append(OrderedList,InterList,ActualList)
vhdl2ges_create_actual_list(CompName,
  [element(vhdl_name(prefix(Assoc),vhdl_range(AssocLow,to,AssocHigh)),
           vhdl_name(prefix(Name),vhdl_range(NameLow,to,NameHigh)))
     |ElementList],ActualList) :-
  !,
  vhdl2ges_expand_name(Assoc,AssocHigh,AssocLow,AssocList),
  vhdl2ges_expand_name(Name,NameHigh,NameLow,NameList),
  vhdl2ges_check_alias(NameList,RealList),
  vhdl2ges_assert_element(AssocList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,InterList),
  comp(CompName,SignalList),
  vhdl2ges_retract_element(SignalList,OrderedList),
  append(OrderedList,InterList,ActualList).
vhdl2ges_create_actual_list(CompName,
  [element(Assoc,
           vhdl_name(prefix(Name),vhdl_range(NameHigh,downto,NameLow)))
     |ElementList],ActualList) :-
  !,
  vhdl2ges_comp_vector_table(CompName,Assoc,AssocList),
  vhdl2ges_expand_name(Name,NameHigh,NameLow,NameList),
  vhdl2ges_check_alias(NameList,RealList),
  vhdl2ges_assert_element(AssocList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,InterList),
  comp(CompName,SignalList),
  vhdl2ges_retract_element(SignalList,OrderedList),
  append(OrderedList,InterList,ActualList).
vhdl2ges_create_actual_list(CompName,
  [element(Assoc,
           vhdl_name(prefix(Name),vhdl_range(NameLow,to,NameHigh)))
     |ElementList],ActualList) :-
  !,
  vhdl2ges_comp_vector_table(CompName,Assoc,AssocList),
```

```
  vhdl2ges_expand_name(Name,NameHigh,NameLow,NameList),
  vhdl2ges_check_alias(NameList,RealList),
  vhdl2ges_assert_element(AssocList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,InterList),
  comp(CompName,SignalList),
  vhdl2ges_retract_element(SignalList,OrderedList),
  append(OrderedList,InterList,ActualList).
vhdl2ges_create_actual_list(CompName,
  [element(vhdl_name(prefix(Assoc),vhdl_range(AssocLow,to,AssocHigh)),
          Name)|ElementList],ActualList) :-
  !,
  vhdl2ges_expand_name(Assoc,AssocHigh,AssocLow,AssocList),
  vhdl2ges_vector_table(Name,NameList),
  vhdl2ges_check_alias(NameList,RealList),
  vhdl2ges_assert_element(AssocList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,InterList),
  comp(CompName,SignalList),
  vhdl2ges_retract_element(SignalList,OrderedList),
  append(OrderedList,InterList,ActualList).
vhdl2ges_create_actual_list(CompName,
  [element(vhdl_name(prefix(Assoc),vhdl_range(AssocHigh,downto,AssocLow)),
          Name)|ElementList],ActualList) :-
  !,
  vhdl2ges_expand_name(Assoc,AssocHigh,AssocLow,AssocList),
  vhdl2ges_vector_table(Name,NameList),
  vhdl2ges_check_alias(NameList,RealList),
  vhdl2ges_assert_element(AssocList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,InterList),
  comp(CompName,SignalList),
  vhdl2ges_retract_element(SignalList,OrderedList),
  append(OrderedList,InterList,ActualList).
vhdl2ges_create_actual_list(CompName,
  [element(vhdl_name(prefix(Assoc),[AssocNumb],[]),
          vhdl_call(Name,[element(null,NameNumb)]))
     |ElementList],ActualList) :-
  !,
  vhdl2ges_expand_name(Assoc,AssocNumb,AssocNumb,AssocList),
  vhdl2ges_expand_name(Name,NameNumb,NameNumb,NameList),
  vhdl2ges_check_alias(NameList,RealList),
  vhdl2ges_assert_element(AssocList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,InterList),
  comp(CompName,SignalList),
  vhdl2ges_retract_element(SignalList,OrderedList),
  append(OrderedList,InterList,ActualList).
vhdl2ges_create_actual_list(CompName,
  [element(Assoc,
          vhdl_call(Name,[element(null,NameNumb)]))
     |ElementList],ActualList) :-
  !,
  vhdl2ges_expand_name(Name,NameNumb,NameNumb,NameList),
  vhdl2ges_check_alias(NameList,RealList),
```

```
  vhdl2ges_assert_element([Assoc],RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,InterList),
  comp(CompName,SignalList),
  vhdl2ges_retract_element(SignalList,OrderedList),
  append(OrderedList,InterList,ActualList).
vhdl2ges_create_actual_list(CompName,
  [element(vhdl_name(prefix(Assoc),[AssocNumb],[]),
           Name)
     |ElementList],ActualList) :-
  !,
  vhdl2ges_expand_name(Assoc,AssocNumb,AssocNumb,AssocList),
  vhdl2ges_check_alias([Name],RealList),
  vhdl2ges_assert_element(AssocList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,InterList),
  comp(CompName,SignalList),
  vhdl2ges_retract_element(SignalList,OrderedList),
  append(OrderedList,InterList,ActualList).
vhdl2ges_create_actual_list(CompName,
    [element(Assoc,Name)|ElementList],ActualList) :-
  vhdl2ges_vector_table(Name,NameList),
  !,
  vhdl2ges_comp_vector_table(CompName,Assoc,AssocList),
  vhdl2ges_check_alias(NameList,RealList),
  vhdl2ges_assert_element(AssocList,RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,InterList),
  comp(CompName,SignalList),
  vhdl2ges_retract_element(SignalList,OrderedList),
  append(OrderedList,InterList,ActualList).
vhdl2ges_create_actual_list(CompName,
    [element(Assoc,Name)|ElementList],ActualList) :-
  !,
  vhdl2ges_check_alias([Name],RealList),
  vhdl2ges_assert_element([Assoc],RealList),
  vhdl2ges_create_actual_list(CompName,ElementList,InterList),
  comp(CompName,SignalList),
  vhdl2ges_retract_element(SignalList,OrderedList),
  append(OrderedList,InterList,ActualList).
vhdl2ges_create_actual_list(CompName,[_|ElementList],ActualList) :-
  !,
  vhdl2ges_create_actual_list(CompName,ElementList,ActualList).
```

The Prolog routine **vhdl2ges_check_alias/2** is called by **vhdl2ges_create_actual_list/3**. The Prolog routine **vhdl2ges_check_alias/2** checks that the signal is not an alias and if so, returns the actual name for the alias name.

```
vhdl2ges_check_alias([],[]) :- !.
vhdl2ges_check_alias([Alias|Tail],[Actual|ActualList]) :-
  vhdl2ges_alias_table(Alias,Actual),
  !,
  vhdl2ges_check_alias(Tail,ActualList).
vhdl2ges_check_alias([Actual|Tail],[Actual|ActualList]) :-
  !,
  vhdl2ges_check_alias(Tail,ActualList).
```

vhdl2ges_assert_element/2 is called by **vhdl2ges_create_actual_list/3** to create a temporary table for association lists. The table is related to the *port_map_aspect* of a specific component.

```
vhdl2ges_assert_element([],[]) :- !.
vhdl2ges_assert_element([Assoc|AssocList],[Real|RealList]) :-
  !,
  assert(assoc(Assoc,Real)),
  vhdl2ges_assert_element(AssocList,RealList).
```

vhdl2ges_retract_element/2 is called by **vhdl2ges_create_actual_list/3**. It is used to retract the temporary table for association lists created by **vhdl2ges_assert_element/2**. The temporary table is retracted in the order specified by the *local_port_clause* of the component being *translated*.

```
vhdl2ges_retract_element([],[]) :- !.
vhdl2ges_retract_element([Assoc|AssocList],
    [NewHead|NewTail]) :-
  retract(assoc(Assoc,NewHead)),
  !,
  vhdl2ges_retract_element(AssocList,NewTail).
vhdl2ges_retract_element(_,[]) :- !.
```

The following set of rules is used to generate the GES extraction rules within an output file called **rulefile**. The first Prolog routine, **vhdl2ges_gen_rule**, is called by **vhdl2ges_sub**.

```
vhdl2ges_gen_rule :-
  tell(rulefile),
  component(Name,_),
  write(Name),write(' :-'),nl,
  vhdl2ges_gen_matching_goals_in_order(0,Acc),
  vhdl2ges_gen_unique_rule(Acc),
  vhdl2ges_gen_not_connected,
  vhdl2ges_gen_retract,
  vhdl2ges_gen_anomaly,
  vhdl2ges_gen_assert,
  nl,
  told.
```

vhdl2ges_gen_matching_goals_in_order/2 is called by **vhdl2ges_gen_rule**. An accumulator is used to keep track of the number of components being used as matching goals. The accumulator value is used to make unique X and Y variables for each matching goal.

```
vhdl2ges_gen_matching_goals_in_order(Acc2,Acc) :-
  retract(goal(Name,SigList)),
  !,
  write('  '),write(Name),write('(N'),
  vhdl2ges_write_sig_name(SigList),
  write(',X'),write(Acc2),
  write(',Y'),write(Acc2),
  write(',_),'),nl,
  NewAcc2 is Acc2 + 1,
  vhdl2ges_gen_matching_goals_in_order(NewAcc2,Acc),!,
  assert(goal(Name,SigList)).
vhdl2ges_gen_matching_goals_in_order(Acc,RetAcc) :-
  RetAcc is Acc - 1.
```

A goal in the facts database would be converted in the following manner.

```
| ?- listing(goal).

goal(and_gate,[a,b,out]).
goal(and_gate,[c,d,out2]).

yes
| ?- vhdl2ges_gen_matching_goals_in_order(0,X).
  and_gate(Na,Nb,Nout,X0,Y0,_),
  and_gate(Nc,Nd,Nout2,X1,Y1,_),

X = 1 ;

no
| ?-
```

The value returned in X is one less than the number of goals.

The Prolog routine **vhdl2ges_write_sig_name/1** is called by **vhdl2ges_gen_listing/0**, **vhdl2ges_gen_not_connected/0**, **vhdl2ges_gen_retract/0**, **vhdl2ges_gen_anomaly/0**, **vhdl2ges_gen_assert/0**, and **vhdl2ges_gen_matching_goals_in_order/2**. The Prolog routine **vhdl2ges_write_sig_name/1** takes a signal list and writes the list in the correct order, each signal name with a prefix of "N" (excluding the first signal name in the list).

```
vhdl2ges_write_sig_name([Last]) :-
  !,write(Last).
vhdl2ges_write_sig_name([Sig|List]) :-
  write(Sig),write(',N'),
  !,
  vhdl2ges_write_sig_name(List).
```

An example of **vhdl2ges_write_sig_name/1** is shown in the following.

```
| ?- vhdl2ges_write_sig_name([a,b,c,d]).
a,Nb,Nc,Nd
yes
| ?-
```

The Prolog routine **vhdl2ges_gen_unique_rule/1** is called by **vhdl2ges_gen_rule**. It produces **unique_component/1** for the extraction rule.

```
vhdl2ges_gen_unique_rule(Acc) :-
  write('  unique_component(['),
  vhdl2ges_gen_unique_XYs(Acc),
  write(']),'),nl.
```

vhdl2ges_gen_unique_XYs/1 is called by **vhdl2ges_gen_unique_rule/1**. It produces an ordered listing of X-Y pairs.

```
vhdl2ges_gen_unique_XYs(0) :-
  !,
  write('[X0,Y0]').
vhdl2ges_gen_unique_XYs(Acc) :-
  write('[X'),
  write(Acc),
  write(',Y'),
  write(Acc),
  write('],'),
  NewAcc is Acc - 1,
  !,
  vhdl2ges_gen_unique_XYs(NewAcc).
```

The following is an example of the output produced by **vhdl2ges_gen_unique_XYs/1**.

```
| ?- vhdl2ges_gen_unique_XYs(5).
[X5,Y5],[X4,Y4],[X3,Y3],[X2,Y2],[X1,Y1],[X0,Y0]
yes
| ?-
```

The Prolog routine **vhdl2ges_gen_not_connected/0** is called by **vhdl2ges_gen_rule**. The Prolog routine **vhdl2ges_gen_not_connected/0** produces the **not_connected/2** goal used by the extraction rule.

```
vhdl2ges_gen_not_connected :-
  internal_sig([]),
  !.
vhdl2ges_gen_not_connected :-
  !,
  internal_sig(List1),
  external_sig(List2),
  write('  not_connected([N'),
  vhdl2ges_write_sig_name(List1),write('],[N'),
  vhdl2ges_write_sig_name(List2),write(']),'),nl.
```

The Prolog routine **vhdl2ges_gen_retract/0** is called by **vhdl2ges_gen_rule**.

```
vhdl2ges_gen_retract :-
  retract(goal(Name,SigList)),
  !,
  write('  retract('),write(Name),write('(N'),
  vhdl2ges_write_sig_name(SigList),
  write(',_,_,_)),'),nl,
  vhdl2ges_gen_retract.
vhdl2ges_gen_retract :- !.
```

The Prolog routine **vhdl2ges_gen_anomaly/0** is called by **vhdl2ges_gen_rule**. The Prolog routine **vhdl2ges_gen_anomaly/0** produces the "find_anomaly" line in the extraction rule.

```
vhdl2ges_gen_anomaly :-
  internal_sig([]),
  !.
vhdl2ges_gen_anomaly :-
  !,
  component(Name,_),
  external_sig(List),
  write('  find_anomaly_list('),write(Name),write('(N'),
  vhdl2ges_write_sig_name(List),
  write(',X0,Y0,1),[N'),
  internal_sig(List2),
  vhdl2ges_write_sig_name(List2),
  write(']),'),nl.
```

vhdl2ges_gen_assert/0, is called by **vhdl2ges_gen_rule**. **vhdl2ges_gen_assert/0** is used to generate the assert line of the extraction rule.

```
vhdl2ges_gen_assert :-
  component(Name,_),
  external_sig(List),
  write('  assert('),write(Name),write('(N'),
  vhdl2ges_write_sig_name(List),
  write(',X0,Y0,1)),'),nl,write('  fail.'),nl,
  write(Name),write('.'),!.
```

The following Prolog routines generate a **find_anomaly/2** rule for GES. The output is placed in a file called **anomalyfile**. The Prolog routine **vhdl2ges_gen_find_anomaly/0** is called by **vhdl2ges_sub/0**.

```
vhdl2ges_gen_find_anomaly :-
  tell(anomalyfile),
  vhdl2ges_anomaly_first_line,
  component(Name,_),
  external_sig(List),
  vhdl2ges_anomaly_gen_match(Name,List,List),
  vhdl2ges_anomaly_gen_tail(Name),
  told.
```

vhdl2ges_anomaly_first_line/0, is called by vhdl2ges_gen_find_anomaly/0.

```
vhdl2ges_anomaly_first_line :-
  !,
  write('find_anomaly(Comp,Node) :-'),
  nl,
  write('  (').
```

vhdl2ges_anomaly_gen_match/2, is called by vhdl2ges_gen_find_anomaly/0.

```
vhdl2ges_anomaly_gen_match(Name,List1,[Head]) :-
  !,
  write(Name),write('('),
  vhdl2ges_anomaly_node_blank(List1,[Head]),
  write('X,Y,_)),'),nl.
vhdl2ges_anomaly_gen_match(Name,List1,[Head|List2]) :-
  write(Name),write('('),
  vhdl2ges_anomaly_node_blank(List1,[Head|List2]),
  write('X,Y,_);'),nl,write('  '),
  !,
  vhdl2ges_anomaly_gen_match(Name,List1,List2).
```

vhdl2ges_anomaly_node_blank/2 is called by vhdl2ges_anomaly_gen_match/2.

```
vhdl2ges_anomaly_node_blank([],_) :- !.
vhdl2ges_anomaly_node_blank([Head|List1],[Head|List2]) :-
  write('Node,'),
  !,
  vhdl2ges_anomaly_node_blank(List1,[Head|List2]).
vhdl2ges_anomaly_node_blank([_|List1],List2) :-
  write('_,'),
  !,
  vhdl2ges_anomaly_node_blank(List1,List2).
```

vhdl2ges_anomaly_gen_tail/1 is called by vhdl2ges_gen_find_anomaly/0.

```
vhdl2ges_anomaly_gen_tail(Name) :-
  write('  write(''Failure extracting component ''),write(Comp),'),nl,
  write('  write(''.''),nl,write(''  Internal node, ''),write(Node),'),nl,
  write('  write('', connected to '),write(Name),
  write(' at X:''),'),nl,
  write('  write(X),write('', Y:''),write(Y),write(''.''),nl.'),nl.
```

The Prolog routines **vhdl2ges_gen_rule_name/0** and **vhdl2ges_gen_listing/0** are both called by **vhdl2ges_sub/0**. Both routines generate the appropriate Prolog code for GES to call the extraction rule and call the list rule.

```
vhdl2ges_gen_rule_name :-
  tell(rulelistfile),
  component(Name,_),
  write(Name),write(','),
  nl,
  told.

vhdl2ges_gen_listing :-
  tell(listlistfile),
  component(Name,_),
  write('list_'),write(Name),
  write(','),nl,
  told,
  fail.
vhdl2ges_gen_listing :-
  !,
  tell(listfile),
  component(Name,_),
  external_sig(List),
  write('list_'),write(Name),write(' :-'),nl,
  write('  retract('),write(Name),write('(N'),
  vhdl2ges_write_sig_name(List),write(',X,Y,T)),'),nl,
  write('  write('),write(Name),write('(N'),
  vhdl2ges_write_sig_name(List),write(',X,Y,T)),'),
  write('  write(''.''),nl,'),nl,
  write('  fail.'),nl,
  write('list_'),write(Name),write('.'),nl,
  told.
```

4.2 Building VHDL2GES

As stated earlier, ***vhdl2ges*** uses the VHDL intermediate form generated by *vhdl_parser*. Therefore, ***vhdl2ges*** must be compiled into a Prolog saved state of the *vhdl_parser*. The following Quintus Prolog session demonstrates building and saving *vhdl2ges*.

```
% vhdl_parser

Quintus Prolog Release 2.4 (VAX, Ultrix 2.0-2.2)
Copyright (C) 1988, Quintus Computer Systems, Inc.  All rights reserved.
1310 Villa Street, Mountain View, California  (415) 965-7700

| ?- compile(vhdl2ges).
[compiling /usr/users/ges/src/pro/vhdl2ges/vhdl2ges.pl...]
[Undefined procedures will just fail ('fail' option)]
[vhdl2ges.pl compiled 13.433 sec 12,752 bytes]

yes
| ?- save(vhdl2ges).
[ Prolog state saved into /usr/users/ges/src/pro/vhdl2ges ]

yes
| ?-
```

4.3 Executing VHDL2GES

There are two steps to executing ***vhdl2ges***. The Prolog saved state is called at the system prompt. Afterwards, the Prolog routine **vhdl2ges/1** is called at the Prolog prompt. A short session demonstrating how this is done follows.

```
% vhdl2ges

yes
| ?- vhdl2ges(foo).
**
%
```

The VHDL file for this example may be called either **foo.vhd** or **foo.vhdl**.

There are five files produced by ***vhdl2ges***. These files are called **anomalyfile**, **listfile**, **listlistfile**, **rulefile**, and **rulelistfile**. The file **anomalyfile** contains a **find_anomaly/2** Prolog routine for the component being extracted. The file **listfile** contains the Prolog routine for correctly listing the extracted component to the standard output. The file **listlistfile** contains the Prolog call for the Prolog list routine. The file **rulefile** contains the extraction rule for the component. Finally, the file **rulelistfile** contains the Prolog call for the extraction rule. For an example of the information produced by ***vhdl2ges***, let us consider the following VHDL description of a full adder:

```
entity full_Adder is
  port (
      X, Y, Cin: in bit;
      Sum, Cout: out bit
     );

  end full_Adder;

architecture full_adder of full_Adder is

         component nand_gate
            generic(constant tPLH:TIME:=0 ns;
                    constant tPHL:TIME:=0 ns);
             port  (signal A:in bit;
                    signal B:in bit;
                    signal C:out bit);
            end component;

         for all : nand_gate use entity work.nand_gate( nand_gate );

         component half_adder_cc
          port (
              X, Y     : in bit;
              Sum, Cbar: out bit
             );
            end component;

         for all : half_adder_cc use entity work.half_adder_cc( gate_level );

        signal P, Q, R : bit;

      begin
         C1: NAND_GATE port map (Q, R, Cout);
         C2: HALF_ADDER_CC port map (X, Y, P, Q);
         C3: HALF_ADDER_CC port map (P, Cin, Sum, R);
    end full_adder;
```

vhdl2ges is run to transform the full adder VHDL model.

```
% vhdl2ges

yes
| ?- vhdl2ges(full_adder).
**
%
```

The contents of **anomalyfile** are

```
find_anomaly(Comp,Node) :-
  (full_adder(Node,_,_,_,_,_,X,Y,_);
  full_adder(_,Node,_,_,_,X,Y,_);
  full_adder(_,_,Node,_,_,X,Y,_);
  full_adder(_,_,_,Node,_,X,Y,_);
  full_adder(_,_,_,_,Node,X,Y,_)),
  write('Failure extracting component '),write(Comp),
  write('.'),nl,write('  Internal node, '),write(Node),
  write(', connected to full_adder at X:'),
  write(X),write(', Y:'),write(Y),write('.'),nl.
```

The contents of **listfile** are

```
list_full_adder :-
  retract(full_adder(Nx,Ny,Ncin,Nsum,Ncout,X,Y,T)),
  write(full_adder(Nx,Ny,Ncin,Nsum,Ncout,X,Y,T)),  write('.'),nl,
  fail.
list_full_adder.
```

The contents of **listlistfile** are

```
list_full_adder,
```

The contents of **rulefile** are

```
full_adder :-
  nand_gate(Nq,Nr,Ncout,X0,Y0,_),
  half_adder_cc(Nx,Ny,Np,Nq,X1,Y1,_),
  half_adder_cc(Np,Ncin,Nsum,Nr,X2,Y2,_),
  unique_component([[X2,Y2],[X1,Y1],[X0,Y0]]),
  not_connected([Np,Nq,Nr],[Nx,Ny,Ncin,Nsum,Ncout]),
  retract(half_adder_cc(Np,Ncin,Nsum,Nr,_,_,_)),
  retract(half_adder_cc(Nx,Ny,Np,Nq,_,_,_)),
  retract(nand_gate(Nq,Nr,Ncout,_,_,_)),
  find_anomaly_list(full_adder(Nx,Ny,Ncin,Nsum,Ncout,X0,Y0,1),[Np,Nq,Nr]),
  assert(full_adder(Nx,Ny,Ncin,Nsum,Ncout,X0,Y0,1)),
  fail.
full_adder.
```

The contents of **rulelistfile** are

```
full_adder,
```

4.4 A C-Shell Routine for Building a Customized GES

A C-Shell routine was written so that a series of VHDL files could be read in and a customized GES built from the resulting extraction rules. The following is a portion of the C-Shell routine that pertains to *vhdl2ges*.

```
PATH=/bin:/usr/bin:/usr/local/bin:/usr/users/ges/bin:/usr/users/cad/bin

anomaly=/tmp/anomaly.$$
rules=/tmp/rules.$$
rulelist=/tmp/rulelist.$$
listfile=/tmp/listfile.$$
listlistfile=/tmp/listlistfile.$$



case "$1" in
    -v)
        >$anomaly
        >$rules
        >$rulelist
        >$listfile
        >$listlistfile
        for i in $*
            do
                if test "$i" != -v
                  then
                    if test -f "$i".vhd -o -f "$i".vhdl
                      then
                        (echo 'vhdl2ges('; echo $i ;echo ').') | vhdl2ges
                        cat anomalyfile >> $anomaly
                        rm anomalyfile
                        cat rulefile >> $rules
                        rm rulefile
                        cat rulelistfile >> $rulelist
                        rm rulelistfile
                        cat listfile >> $listfile
                        rm listfile
                        cat listlistfile >> $listlistfile
                        rm listlistfile
                      else
                        echo $i does not exist.  ges not produced.
                        rm $anomaly
                        rm $rules
                        rm $rulelist
                        rm $listfile
                        rm $listlistfile
                        rm -f ges
                        exit 2
                      fi
                  fi
            done
        cp /usr/users/ges/src/sh/verify/ges1 ges
        cat $rulelist >> ges
        rm $rulelist
        cat /usr/users/ges/src/sh/verify/ges2 >> ges
        cat $listlistfile >> ges
```

```
        rm $listlistfile
        cat /usr/users/ges/src/sh/verify/ges3 >> ges
        cat $rules >> ges
        rm $rules
        cat /usr/users/ges/src/sh/verify/ges4 >> ges
        cat $anomaly >> ges
        rm $anomaly
        cat /usr/users/ges/src/sh/verify/ges5 >> ges
        cat $listfile >> ges
        rm $listfile
        echo "compile(ges). save(ges)." | prolog
        exit 0 ;;

            . . .  other switches  . . .

    esac

echo Must specify switch:
echo verify -v file1 ... fileN - to build customized ges from VHDL
echo verify -g magicfile ------- to execute magic, ext2sim, and ges
echo verify -s simfile --------- to execute ges
echo verify -e simfile --------- to execute ges_error1
echo verify -c ----------------- to clean up files
exit 1
```

The "**$$**" used in the first few lines of the C-shell appends the process number of the current process to the files being opened in the **/tmp** directory. Afterwards, a test is made to ensure that the VHDL files exist. As each VHDL file is processed by *vhdl2ges*, the five files produced by *vhdl2ges* are appended to their counterparts in the **/tmp** directory. Once all of the VHDL files have been processed, the five temporary files kept in the **/tmp** directory are merged with the Prolog code of GES. The customized GES is then compiled in Prolog and saved.

V. Limitations of Hierarchical Extraction Methods

The purpose of this section is to discuss some limitations to hierarchical extraction. After the discussion of the limitations, some suggestions for overcoming these limitations will be discussed. The problem specifically involves the types of component configurations that are recognized and extracted. Consider the following example.

Assume that an extraction rule exists for identifying an **AND** gate formed from a **NAND** gate followed by an inverter. A typical GES extraction rule for identifying this configuration is the following.

```
and_gate :-
  nand_gate(Nx,Ny,Ninterm,X0,Y0,_),
  inv(Ninterm,Noutput,X1,Y1,_),
  unique_component([[X1,Y1],[X0,Y0]]),
  not_connected([Ninterm],[Nx,Ny,Noutput]),
  retract(inv(Ninterm,Noutput,_,_,_)),
  retract(nand_gate(Nx,Ny,Ninterm,_,_,_)),
  find_anomaly_list(and_gate(Nx,Ny,Noutput,X0,Y0,1),[Ninterm]),
  assert(and_gate(Nx,Ny,Noutput,X0,Y0,1)),
  fail.
and_gate.
```

From the extraction rule **and_gate/0**, every **NAND** gate followed by an inverter will be replaced with an **AND** gate.

Consider an additional extraction rule, **half_adder_cc/0**, constructed as follows.

```
half_adder_cc :-
  xor_gate(Nx,Ny,Nsum,X0,Y0,_),
  nand_gate(Nx,Ny,Ncbar,X1,Y1,_),
  inv(Ncbar,Ncarry,X2,Y2,_),
  unique_component([[X2,Y2],[X1,Y1],[X0,Y0]]),
  not_connected([Ncbar],[Nx,Ny,Nsum,Ncarry]),
  retract(inv(Ncbar,Ncarry,_,_,_)),
  retract(nand_gate(Nx,Ny,Ncbar,_,_,_)),
  retract(xor_gate(Nx,Ny,Nsum,_,_,_)),
  find_anomaly_list(half_adder_cc(Nx,Ny,Nsum,Ncarry,X0,Y0,1),[Ncbar]),
  assert(half_adder_cc(Nx,Ny,Nsum,Ncarry,X0,Y0,1)),
  fail.
half_adder_cc.
```

We will use the **and_gate/0** and **half_adder_cc/0** rules to perform extraction on the following components.

```
nand_gate(na,nb,ncbar,1,1,1).
xor_gate(na,nb,nsum,10,1,1).
inv(ncbar,ncarry,1,10,1).
```

If the **half_adder_cc/0** extraction rule is used before the **and_gate/0** extraction rule, the following will result.

```
half_adder_cc(na,nb,nsum,ncarry,10,1,1).
```

However, if the **and_gate/0** extraction rule is used before the **half_adder_cc/0** extraction rule, the following will result.

```
xor_gate(na,nb,nsum,10,1,1).
and_gate(na,nb,ncarry,1,1,1).
```

There are three methods for solving this problem. The first method involves ordering the rules such that the **half_adder_cc/0** extraction rule is called before the **and_gate/0** extraction rule. This prevents the **and_gate/0** extraction rule from interfering with proper extraction of the **half_adder_cc/0** extraction rule. The next two methods are interrelated.

If higher-level structural VHDL descriptions are not making use of the fact that an and_gate VHDL description exists, then eliminate the and_gate VHDL description. However, if it is necessary to have an and_gate VHDL description, ensure that all higher-level structural VHDL descriptions take advantage of the and_gate VHDL description. This type of reasoning may require the designer to make more prudent use of VHDL-based models. However, the designer may employ another method of enriching the extraction rule set by simply adding an additional VHDL description for half adders using **AND** and **Exclusive-OR** gates.

The previously described problem is not the only case where hierarchical extraction may require some manual intervention. Some extraction rules might force extractions over component boundaries. An example of how this might occur follows.

The previously defined extraction rules, **and_gate/0** and **half_adder_cc/0**, will be used. Assume a new extraction rule for **half_adder/0**.

```
half_adder :-
  half_adder_cc(Nx,Ny,Nsum,Ncbar,X0,Y0,_),
  inv(Ncbar,Ncout,X1,Y1,_),
  unique_component([[X1,Y1],[X0,Y0]]),
  not_connected([Ncbar],[Nx,Ny,Nsum,Ncout]),
  retract(inv(Ncbar,Ncout,_,_,_)),
  retract(half_adder_cc(Nx,Ny,Nsum,Ncbar,_,_,_)),
  find_anomaly_list(half_adder(Nx,Ny,Nsum,Ncout,X0,Y0,1),[Ncbar]),
  assert(half_adder(Nx,Ny,Nsum,Ncout,X0,Y0,1)),
  fail.
half_adder.
```

The component list from earlier in the section will be used. For clarity, the same netlist is shown below.

```
nand_gate(na,nb,ncbar,1,1,1).
xor_gate(na,nb,nsum,10,1,1).
inv(ncbar,ncarry,1,10,1).
```

If the extraction is performed in the order **and_gate/0**, **half_adder_cc/0**, and **half_adder/0**, the result will be the following.

```
xor_gate(na,nb,nsum,10,1,1).
and_gate(na,nb,ncarry,1,1,1).
```

The three methods presented earlier are also useful in solving this problem.

There is one type of extraction problem that requires greater consideration. Assume we define a four-input **AND** gate as shown in Figure 1. Using the new extraction rule for a four-input **AND** gate, we will extract the circuit shown in Figure 2. Once the extraction process has completed, two different interpretations may result. In one case, the extraction process might yield two four-input **AND** gates and one two-input **AND** gate. In the second case, the extraction process might yield one four-input **AND** gate and four two-input **AND** gates. Currently, the method for solving this problem is to exclude VHDL descriptions for models that contain homogeneous structure.

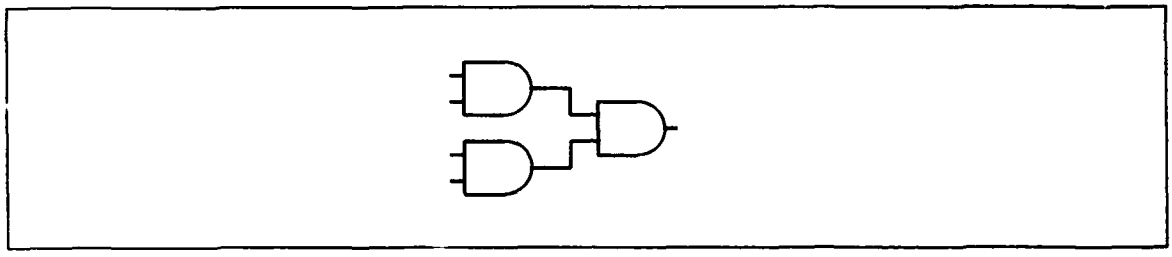

Figure 1. Four-Input **AND** Gate.

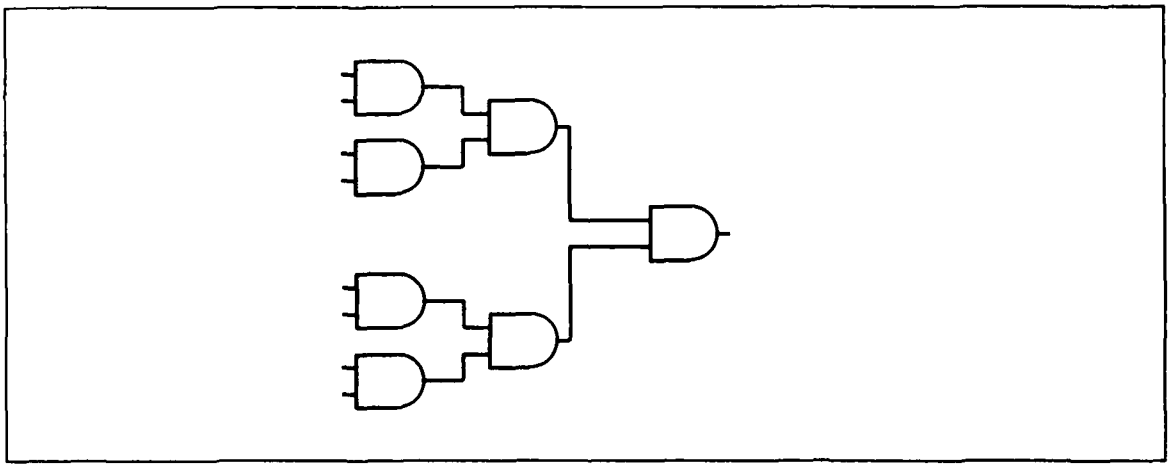

Figure 2. Simple Circuit.

VI. Conclusions

Several items were discussed in this report. A method for generating Prolog extraction rules for GES was provided. Modifications to the general GES extraction rule were also presented. Limitations regarding unguided hierarchical extraction were examined. Methods for overcoming the limitations were offered.

Further work is under way to automate recognition and correction of the limitations to unguided hierarchical extraction. Other additions to the GES environment are also progressing. Future additions include determining propagation delays for gate-level components from *magic* [4] layout descriptions, "pin-to-pin" delay analysis (to include finding critical paths in *magic* layout descriptions), highlighting critical paths in *magic*, back annotation of VHDL models with critical paths, and an expert system extension for generating VHDL models of undocumented components.

References

1. IEEE, Computer Society Standards Committee, "IEEE Standard VHDL Language Reference Manual," *ANSI/IEEE Std 1076-1987*, New York: IEEE Press; 1987.
2. Reintjes, Peter B., "A VHDL Parser in Prolog," Technical Report, Research Triangle Park, North Carolina, 1990.
3. Dukes, Michael A., Frank M. Brown, and Joanne E. DeGroat, *A Generalized Extraction System for VLSI*, July 1987-August 1990, WRDC Technical Report, WRDC-TR-90-5021, Wright Research and Development Center, Wright-Patterson AFB OH, August 1990.
4. Clocksin, W. F. and C. S. Mellish, *Programming in Prolog*, New York: Springer-Verlag; 1987.
5. University of California, Berkeley, Berkeley Distribution of Design Tools, Computer Science Division, EECS Department, University of California at Berkeley; 1986.
6. Quintus Computer Systems, Inc, *Quintus Prolog Library Manual*, Mountain View, California, p. 68, 1988.

U.S. Government printing office 548-074