VLSI Design
1994, Vol. 1, No. 2, pp. 113–126
Reprints available directly from the publisher
Photocopying permitted by license only

A VHDL Based Expert System for Hardware Synthesis

SAJJAN G. SHIVA
Computer Science Department, University of Alabama in Huntsville, Huntsville, Alabama

JUDIT U. JONES
Rockwell International, 555 Discovery Drive, Huntsville, Alabama

(Received May 9, 1989, Revised October 16, 1989)

This paper describes an expert system for Hardware Synthesis. Details of the target digital system are input to the expert system using Very High Speed Integrated Circuit Hardware Description Language (VHDL). The VHDL representation is first translated to a knowledge representation scheme known as a 'hologram' which is a combination of rule, frame and semantic network representation schemes. The hologram representation of the target system is then input to the inference engine, which matches the target system to the Knowledge Base components and selects an appropriate set for implementation, and connects them creating a digital circuit. Some design examples are described. The expert system approach results in designs very close to designs from a human designer. In its present form, the system does not perform a design space exploration for alternate designs, but expects the designer to alter the VHDL representation, after observing the results from previous design cycles.

Key Words: *Expert system; Automatic synthesis; VHDL; Knowledge representation; Hardware synthesis*

Due to the complexity of very large scale integrated circuits, the design phase of a new product could take several years of work by design teams. Yet, the time from the conception of a new product to its appearance on the market has to be kept to a minimum to bring new products to market ahead of competition. Design aids such as wire routers, computer aided design systems, simulation programs, and hardware synthesizers have been developed to make the design time shorter. Overviews of hardware synthesis can be found in [1, 2].

In 1978 the Department of Defense initiated the Very High Speed Integrated Circuit (VHSIC) Program, to aid in the production of military integrated circuits. VHDL was developed to help create an integrated design environment, taking the designer through all phases of development, testing, and evaluation. VHDL also facilitated the communication of design requirements between the government and contractors, since operations performed by an integrated circuit (IC) could be described precisely in VHDL. VHDL became the U.S. Government standard hardware description language in 1987 [3–5].

In hardware synthesis tools such as VHDL Design System [6] by IBM and the MIMOLA synthesis system [7] with a VHDL front end tool [8], the high level behavioral VHDL description is first translated into a register transfer (RT) level representation, which is then translated into hardware.

The IBM VHDL Design System is similar to the VHDL design environment described in the next Section. Before synthesis the VHDL structure is "flattened" by the Simplifier. The Synthesizer first creates a technology independent block structured logic model which is composed of latch elements, combinatorial elements and predefined components (given by component instantiations). Then the design is converted into a technology dependent logic model, which is used to create test patterns, placement, wiring, chip masks or printed circuit boards.

The MIMOLA system was developed for the design of processors. The VHDL front end tool creates a process graph from the behavioral VHDL representation. This process graph is optimized using com-

Send all Correspondence to: Dr. Sajjan G. Shiva, Computer Science Department, University of Alabama in Huntsville, Huntsville, Alabama 35899, (205) 895-6160

piler optimization techniques to parallelize sequential representations. The VHDL front end tool decomposes and optimizes (globally) the VHDL behavioral representation and serves as an interface between VHDL and the MIMOLA hardware description language. The MIMOLA system develops the hardware and the microcode, and performs design analysis. The input to the synthesis system is a MIMOLA algorithm and a set of constraints. The algorithm and the constraints are user modifiable to develop several versions of the hardware designed. The input is transformed into intermediate files, then the algorithmic programs are mapped to RT-behavior level programs. Up to three different versions can be generated depending on variable bindings. This is followed by the decomposition of complex statements with the help of some heuristics, with an attention paid to common subexpressions. This is followed by statement scheduling to maximize parallelization (for microinstructions), register assignment, module selection (arithmetic modules), generating interconnections and control.

VHDL has also been utilized in the automated design of control structures. The Control Synthesis System in [9] designs a Finite State Automaton Controller from a VHDL instruction set and a microarchitecture. This controller is then translated into a physical implementation by commercially available CAD tools such as Datapath Compiler by VTItools.

Research in the area of developing an expert system which would function as a design tool in the VHDL design environment has been done at the University of Alabama in Huntsville (UAH) by Green [2, 10], Klon [11], and Jones [12]. Green designed the University of Alabama Hardware Expert Synthesis System (UHESS) which serves as a design consultant in the selection of VHSIC ICs. Klon investigated the issues concerning interfacing UHESS to VHDL. He extended Green's knowledge base representation scheme to accommodate the module and signal concepts of VHDL and named it a hologram. Jones extended the hologram representation and investigated the process of taking a complete circuit design from its VHDL representation and synthesizing it to a finished circuit design. A finished circuit design contains modules (which correspond to existing hardware functions) and their interconnections. This paper describes the current form of the hardware synthesizer.

OVERVIEW OF THE EXPERT SYSTEM

Figure 1 shows the VHDL design environment [13] consisting of the analyzer (a compiler), the design library, the profiler, the simulator, and other design and analysis tools.

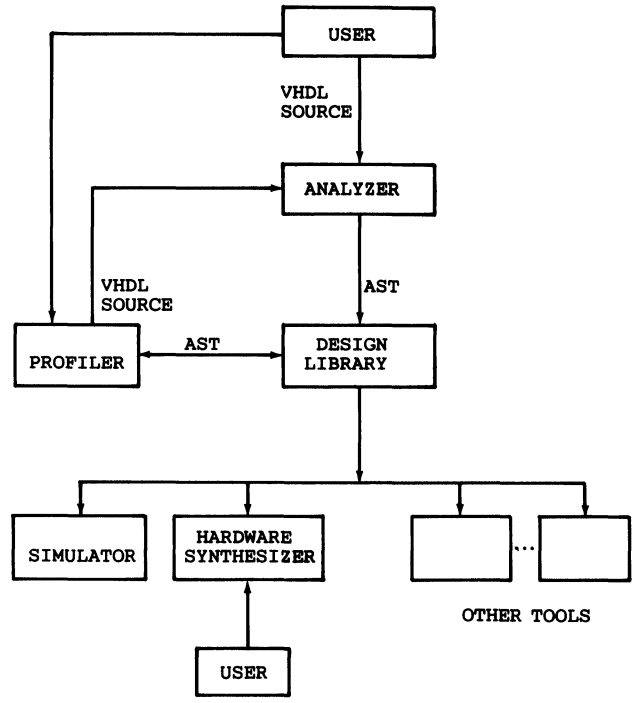


FIGURE 1 VHDL Design Environment.

To develop new hardware in the VHDL design environment, first a VHDL source code description is developed. The VHDL source code specifies the performance requirements of the target hardware system. It can be at one or more levels of a system design hierarchy such as system level, component level, logic level, gate level, or below. The VHDL analyzer translates the source code into an intermediate form known as the abstract syntax tree (AST) which is incorporated into the design library. Based on user instructions, the profiler generates an AST representation which reflects the correct configuration (i.e., entities with the correct revisions are selected). The output of the profiler is incorporated into the design library and presented to the hardware synthesizer tool. The hardware synthesizer then produces a circuit design.

Figure 2 shows the block diagram of the hardware synthesizer. The hardware synthesizer has three major components: the AST to Inference Engine Input (IEI) translator, the inference engine and the knowledge base (KB). Top level details of these components are described in this section. Subsequent sections will provide further details.

AST to IEI Translator

The hardware synthesizer first translates the AST representation to the IEI representation. When por-

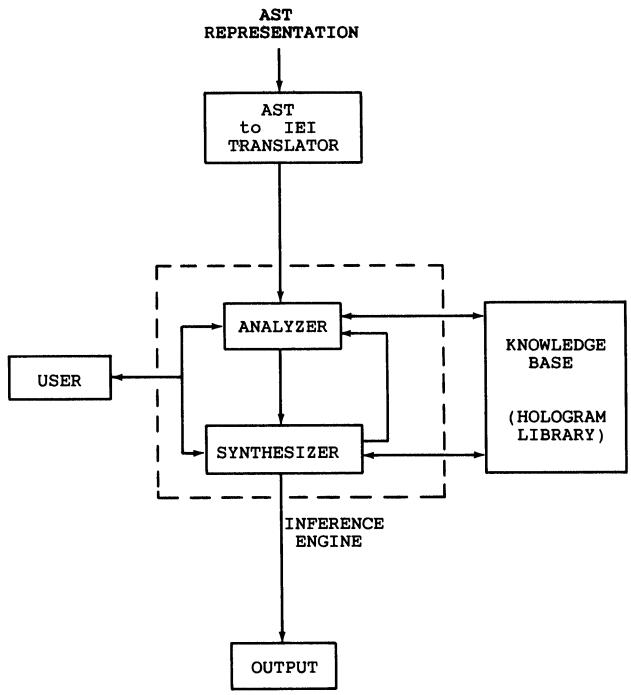


FIGURE 2 The Hardware Synthesizer.

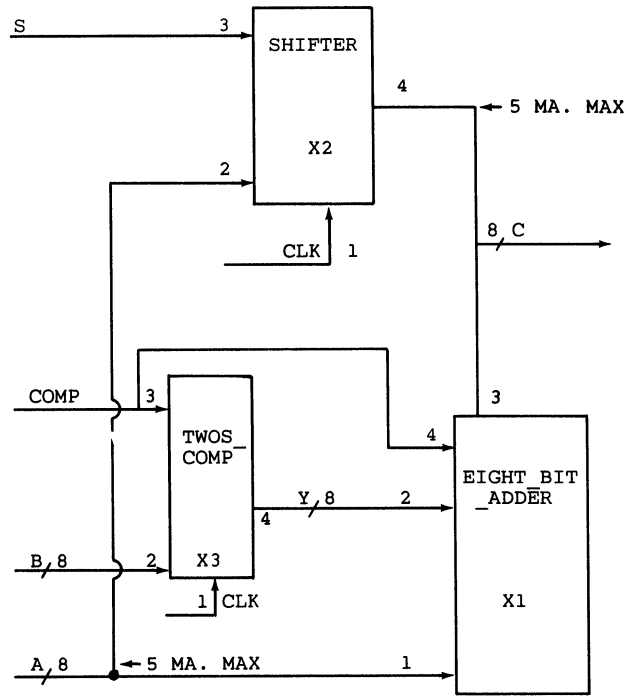


FIGURE 3 Schematic diagram for an ALU. [10]

tions of VHDL representations cannot be translated the user is requested to modify the VHDL representation until a translatable AST representation is reached.

The Knowledge Base

The KB contains holograms of hardware devices. The hologram representation scheme combines frames [14, 15], production rules [14, 15], and semantic networks [14]. The basic knowledge representation mechanism in the hologram scheme is the prototype. A prototype contains frames and rules. The frames contain slots for knowledge storage. This knowledge consists of data pertaining to hardware functions and rules establishing default values and knowledge heuristics. There are two kinds of prototypes in the hologram representation: modules and signals. Modules (a collection of integrated circuits with their interfaces and functions) represent objects, functions, and activities; while signals are carriers of information between modules. Holograms are a network of module and signal prototypes.

As an illustration of the hologram representation, Figure 3 shows a schematic diagram for a simple arithmetic logic unit (ALU) and Figure 4 presents corresponding abbreviated module and signal prototypes.

The module prototypes in Figure 4 contain attributes and a function description. The values of the attributes uniquely describe a module. This module has three subfunctions. The function description contains a detailed description of the operations performed by the module. Port names (input, output, and local) are paired with their typing identifier which relates to its signal prototype (for example (A, BUS)). Local signals are internal to a module. For example, Y is a local signal in Figure 4. Element assignments pair a circuit element number with the name of the component (for example (1, EIGHT_BIT_ADDER)). The component relates to its module prototype. The netlists are listings of pairs of circuit element numbers and their port numbers. For example, signal S is connected to port 3 of module 2, the SHIFTER. The element port pairs form a linked list, and this list is connected to the identifier of the signal which is connected to it. The heuristics slot references usage rules.

For the BUS signal prototype of Figure 4 the descriptor describes the bus as eight BIT. The type identifier itself is a link to the BIT signal structure which is a predefined terminal. Each module and signal structure is similarly expanded until a terminal structure is reached.

Terminal elements of the prototype structures are predefined. Terminal elements do not reference other prototypes. The modifier slot entry is filled in with "terminal." An example for the terminal prototype is signal BIT in Figure 4.

Figure 5 presents the hologram structure for the

```
FUNCTION NAME: ARITHMETIC LOGIC UNIT
TYPE : MODULE
MODULE NUMBER: 0
ATTRIBUTES : ALU,ADD,SHIFT,8-BIT,LS
NUMBER OF SUBFUNCTIONS: 3
FUNCTION DESCRIPTION:
BEGIN
      FUNCTION 1:
      FUNCTION NAME: LEFT_SHIFT_ZERO_FILL
      ATTRIBUTES : POSITIVE_EN,POSITIVE_EDGE_TRIG,3_STATE
      BEGIN
      IF (S AND CLK) THEN LSZF(A) --> C
                     ELSE    Z    --> C
      END IF
      END

      FUNCTION 2:
      FUNCTION NAME: TWOS_COMP
      ATTRIBUTES : POSITIVE_EN,POSITIVE_EDGE_TRIG
      BEGIN
      IF (COMP AND CLK) THEN TCOMP(B) --> Y
      END IF
      END

      FUNCTION 3:
      FUNCTION NAME: ADD
      ATTRIBUTES : POSITIVE_EN,3_STATE
      BEGIN
      ADD(A,Y) --> ADDER_OUT
      IF COMP THEN ADDER_OUT --> C
             ELSE      Z     --> C
      END IF
      END
END
IN:  A, BUS
     B, BUS
     S, BIT
     CLK, BIT
     COMP, BIT
OUT:  C, BUS
LOCAL:  Y, BUS
ELEMENT ASSIGNMENTS:
      1, EIGHT_BIT_ADDER
      2, SHIFTER
      3, TWOS_COMP
NETLISTS:
      A; (1,1), (2,2)
      B; (3,2)
      S; (2,3)
      CLK; (2,1), (3,1)
      COMP;  (1,4), (3,3)
         C;  (1,3), (2,4)
         Y;  (1,2), (3,4)
   HEURISTICS:  RULE071
                RULE076
   ...


NAME:  EIGHT_BIT_ADDER
TYPE:  MODULE
MODULE NUMBER: 1
...
...


NAME:  SHIFTER
TYPE:  MODULE
MODULE NUMBER: 2
...
...

NAME:  TWOS_COMP
TYPE:  MODULE
MODULE NUMBER: 3
...
...

NAME:  BUS
TYPE:  SIGNAL
DESCRIPTOR:  8, BIT
CONDITIONS:  5, MA., MAX
HEURISTICS:  RULE062
RULE065

NAME:  BIT
TYPE:  SIGNAL
MODIFIER:  TERMINAL
```

FIGURE 4 Abbreviated hologram for the ALU

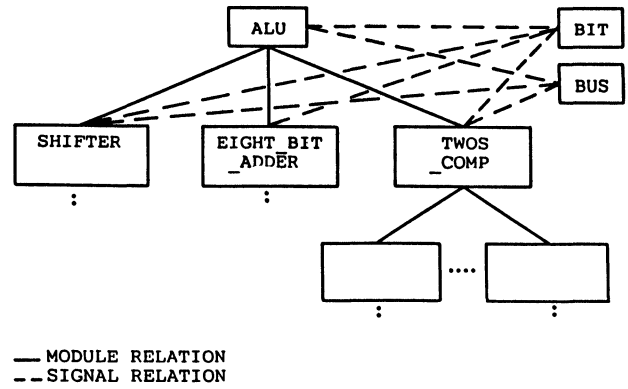


FIGURE 5 Abbreviated hologram structure for an ALU. [10]

ALU example. The solid lines represent relations to modules and the broken lines, relations to signals. The tree hierarchy of modules represent the structural decomposition of a diagonal circuit. Signal prototypes represent decomposition of a digital circuit. Signal prototypes represent the semantics of connectivity between them. Each module in the leaf nodes of the module hierarchy is a terminal module.

The KB holograms can be terminal holograms of existing hardware functions or holograms designed earlier, consisting of existing hardware functions. The holograms in the KB may be at a different level of sophistication (detail). They may contain only one gate, a combinational circuit, a complex circuit (such as a printer interface module), or a whole computing system.

The Inference Engine

To understand better the operations needed to be performed by the inference engine, the operations performed by a human during circuit design are examined. The main operations are: decomposition of a specification into existing hardware components, selection of hardware components, interconnection of components to create the system, and evaluation of the finished design. These operations become the following steps for the inference engine:

a. Matching and similarity assessment: Module prototypes are compared to KB holograms to find a matching or similar hologram. This is equivalent to the selection of components.

b. Decomposition: When no matching or similar hologram is found in the KB, module prototypes are decomposed to their components. This is equivalent to the decomposition performed by a human.

c. Name Unification: This process identifies corresponding signal names in modules and selected KB holograms when the name and number of ports do not match. This will help with the connection of components when knowledge is necessary.

d. Evaluation of the design: The aggregate physical properties of the system designed are computed and compared with the specification. This is equivalent to the evaluation performed by a human.

The inference engine has two major blocks: the analyzer and the synthesizer.

The analyzer decomposes the IEI, creating, as a result, a tree hierarchy of modules which are also connected by signals (i.e., the target hologram). At the leaf nodes of this tree, the modules are holograms found in the KB.

Given the IEI, the analyzer begins to search the KB for similar holograms. Attributes of the IEI module and the attributes of the root modules of KB holograms are compared, and *similarity is evaluated.* The outcome of this search could be: a matching hologram found, a similar hologram found, or a similar hologram does not exist. In each case the actions to be taken are as follows:

a. When a matching hologram is found, its properties will be copied into the module after some *name unifications.* If the matching KB hologram has submodules, its subtree is copied also into the module.

b. When a similar hologram is found, the deficiencies of the selected hologram will be analyzed, corrected if possible, and components (submodules) are generated (i.e., the module is decomposed). Correcting the deficiencies is attempted by using heuristics. For example if the selected module has less BITs than the BIT-attribute indicates, then submodules are generated which collectively have at least as many BITs as are needed. If correcting the deficiencies is not possible, then the hologram will be rejected, and the case of no similar hologram will be considered.

c. When a similar hologram does not exist, the module is decomposed to its submodules, if the function description of the IEI module has components. If the decomposition is not possible, then the help of the designer is requested.

The details of the similarity evaluation process are provided in the section (*Similarity Assessment*). When the module is decomposed, each newly created submodule is subject to the above process. This process could yield new submodules, which are in turn subjected to the procedure described above. This process will continue until all submodules generated are processed, and each leaf node corresponds to a matching KB hologram. The phase just described is the requirement decomposition phase.

The synthesizer evaluates the new design, based on the physical properties of the terminal holograms. Here, physical properties of children nodes are synthesized to become the physical parameters of the parent node. This process begins at the leaf nodes and propagates actual design constraints upward in the tree until the root module of the target hologram is reached. These actual design constraints refer to propagation delays, power supply requirements, compatibility of signal drivers and receivers, etc. The physical properties of the new design are compared to the requirements contained in the IEI module.

The difficulty lies in the fact that the design uses nonmonotonic reasoning. The correctness of design assumptions can be verified only by exploring all consequences of assumptions. Assumptions are the attribute evaluating KB heuristics, and heuristics which determine what action to take to improve similarity. The incorrect assumptions must be pinpointed and other design assumptions substituted. If the new design does not meet all the requirements of the IEI module, additional requirements are given by the user (by altering the VHDL representation), and the design process is repeated. This phase of the design may be aided by a design tool facilitating overall system planning described in [16].

KNOWLEDGE REPRESENTATION

In this section first the reasons for the selection of the knowledge representation scheme are given, followed by the detailed description of the knowledge representation.

Three popular knowledge representation schemes are: production rules, frames and semantic nets.

For the representation of modules and signals production rules alone are not suited to represent heterogeneous knowledge grouped together. The problem is the awkward acquisition of knowledge pertaining to a particular device, since production rules are unordered. Frames are good devices to group together heterogeneous knowledge, but they can not easily express special knowledge about the use of the frame. The prototype representation combines the advantages of both these representations.

Comparison of two semantic nets is slow, since the whole network has to be traversed. But new meanings are easily created. Pure prototype representation does not facilitate the easy creation of new meanings; but comparison is easy, since only the root modules and signals need to be compared.

The hologram representation used by the synthesis system holds advantages over pure prototype or pure semantic network representation and provides mechanisms for both the dynamic creation of new meanings and easy comparison.

The attributes of the module prototypes represent the properties of objects in a very compact form. During similarity assessment, attributes of the root module of KB holograms or the absence of attributes is evaluated according to heuristic rules. The use of attributes is beneficial because of three reasons. First, KB search would take a considerable amount of time if each field of the root modules of KB holograms were compared to each field of a module (to be matched) when a similar hologram is searched for. Comparing and evaluating only the attributes of these objects speeds up the similarity assessment process. Second, differences between KB holograms and a module are efficiently determined by the sum of the attribute differences. Without the attributes, analyzing the differences would take considerable search time, knowledge, and storage requirement. Third, optimization for different design options is facilitated by the use of attributes.

There are several requirements for which a digital design could be optimized. For example: number of components, cost, speed, size, heat generated, long life, mass production, etc. To optimize a design to any of these requirements or any combination of them requires a different approach. Accommodating different design criteria is facilitated with the use of attributes, since only the attribute evaluation heuristics need to be changed. The drawbacks of the use of attributes are that they have to be derived from the VHDL representation, and they require storage.

For easy automatic decomposition and name unification, the function description of module prototypes should be in a special form. This special form should allow: (a) decomposition, (b) name unification, and (c) should contain knowledge of existing components.

To develop this special function description, consider the functioning of a digital circuit. Here, the inputs are transformed to intermediate signals by a function and these intermediate signals are transformed to the next level of intermediate signals by another function, and so on until the output signals are generated. Consider the analogy between these functions and primitives of a language. Then a language comes to mind whose primitives are the functions of actual hardware devices, the Actual Device Primitives (ADPs). The ADPs are statements in the Actual Device Language (ADL). If the function description of a module can be expressed in this language, the hardware design is done in theory (at a high level), because it does not take into consideration lower level design constraints (such as propagation delay, power supply requirements, etc.). Each leaf node in the design tree is an ADP, and each intermediate node is an aggregate of ADPs having as many subfunctions as there are submodules. Therefore, each function definition at a higher level is expressed as a number of subfunctions where the subfunctions have subfunctions and so on. ADP examples are described in the section (*The Actual Device Language*).

With the use of ADL in the decomposition or analysis phase, when no matching (or similar) hologram is found in the KB, the module is simply decomposed such that for each subfunction a submodule is allocated. From the function description allocated to submodules, and from the input and output signals of a module, the input and output signals of the submodules can be determined. The local signals of the module are derived also from the same data. The interactions between the allocated submodules are specified by the local signals. The allocated submodules then become independent modules and all signals connected to them including local signals are considered as inputs or outputs.

With the use of ADL, name unification can also be accomplished. For name unification, there needs to be a way to identify which names correspond to each other in the root module of a KB hologram and a module. Simple positional correspondence could be used in port lists (if there is a convention for port ordering) if the number of ports and their functions match. The situation here could be different when a similar KB hologram is selected, since the number of ports and/or their functions may not match. Since each module has a function description generated from ADPs, the two functions can be compared and names falling in the same position textually unified. Therefore, a priori knowledge of the signals belonging to a hologram finally selected are not necessary.

For automatic hardware synthesis, two approaches could be used. One is that the hardware synthesizer develops all alternatives with a nondeterministic design approach; the other is that the hardware device (HD) constraints are taken into consideration early on in the design. Here HD stands for existing hardware functions. It is easily seen that the nondeter-

ministic design approach would take many iterations, and it would provide design alternatives, many of which are not realizable or are very inefficient. With the use of ADL, the nondeterministic design approach is avoided. Therefore, the second approach must be used; some HD constraints have to be taken into consideration at the design specification (VHDL code generation) phase. To enforce these constraints, VHDL representations of non-existing HDs are discovered at the VHDL to ADL translation process. These non-existing HDs then are modified by the designer. This is equivalent to the idea that the purely top down design methodology can not be used, since this would be the nondeterministic approach. A combination of top down and bottom up methodologies is needed.

ADL saves time, since effort is not expended on the design of circuits which will turn out to be unrealizable. When a circuit design is expressed in ADL, the knowledge of available components is implied.

The above discussion shows that the way the specification is written in VHDL has a very large effect on the design implementation.

Similarity Assessment

Figure 6 shows the entity-relationship diagram of the KB. The relationship between a module to be matched and a KB hologram is the similarity. Each module prototype has function attribute (key attribute) which will place it into an HD class (adder, Arithmetic Logic Unit (ALU), processor . . .). The rest of the attributes are divided into two classes: the primary attributes which are associated with the functioning of the HD, and the secondary attributes which express the physical properties of the HD. For example, for an adder the primary attributes would be: number of BITs, full/half, output type, etc., and secondary attributes would be: propagation delay, fanin, fanout, power supply needed, etc.

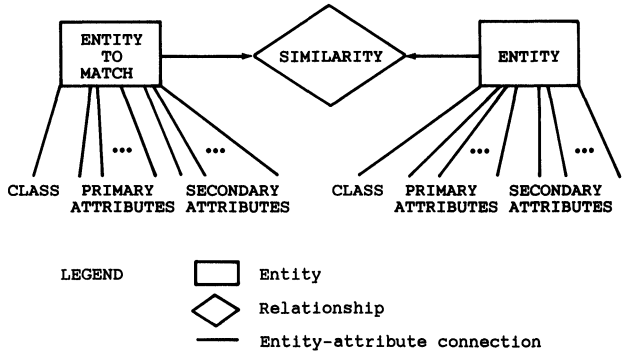


FIGURE 6 The entity-relationship model.

The reason for partitioning the attributes into two classes is that at the decomposition phase, we are trying to find an HD which performs the operations needed. This is a heuristic, and it expresses the preference for functionally correct HDs, since this would give the most efficient design. If for some reason the functionally correct KB hologram does not meet physical requirements, then either another solution is tried, or another hologram is added to the already selected KB hologram. For example, if the selected KB hologram can not drive the inputs of HDs connected to it, a buffer is added (provided this is more efficient than a different approach would be).

At similarity, computational modules are compared to KB holograms with the same key attributes. First, the primary attributes are processed. Based on the value of the attribute of the KB hologram, the value of the same type of attribute of the module, and the associated KB heuristic, a score is generated. (These heuristics are different from the heuristic rules contained in the hologram which were described in the previous section.) The scores of the primary attributes of the root module of the hologram are added together, and a set of holograms with the highest score is selected for further processing. For this set, the secondary attributes are processed. The sum of the secondary attribute scores is generated and added to the sum of primary scores for each hologram in this set. The hologram with the highest composite score is selected. The similarity attribute of the selected hologram is generated, comparing its score to the maximum attainable score.

As an example, Figure 7 shows the selection process using attributes and scores. Here we want to select a 16-BIT full adder. After the processing of the primary attributes, the device families 7483 and 74283 are selected. After the processing of the secondary attributes, the device 74LS283 is selected. The maximum attainable score is 8, because there are 8 attributes to match. The similarity attribute computed is 6.15/8.

ADPs express the general operations of the HD; therefore, properties of HDs which are not given explicitly in the function description are described with attributes. For example, clock information (such as positive edge triggered), output information (such as open collector) and some general electrical characteristics (such as series designation i.e., S, LS, H) for Transistor-Transistor Logic (TTL) devices.

The Knowledge Base

The data model for the KB is a hyper-semantic data model. The uniform handling of knowledge and data is necessary to ensure flexibility of design. Knowl-

ATTRIB-UTE TO MATCH	ADDERS / PRI ATTRIBUTES	80 ATTR	80 SCORE	82 ATTR	82 SCORE	83,283 ATTR	83,283 SCORE
FULL	FULL	FULL	1	FULL	1	FULL	1
16	BIT NUMBER	1	0.06	2	0.12	4	0,25
-	NO OF COMP	SINGL		SINGL		SINGL	
-	EXTRA FUNC	GATED	0.8	-	1	-	1
	SUM		1.86		2.12		2.25

Now devices 83 and 283 are in the set.

ATTRIB-UTE TO MATCH	ADDERS / SEC ATTRIBUTES	SN74LS83A ATTR	SN74LS83A SCORE	SN7283A ATTR	SN7283A SCORE	SN74LS283 ATTR	SN74LS283 SCORE
LS	SERIES	LS	1	-	0.9	LS	1
1	FAN IN	1	1	1	1	1	1
10	FAN OUT	5	0.9	5	0.9	5	0.9
1.5	COST	1.6	0.9	1.7	0.8	1.4	1
	SUM		6.05		5.85		6.15

MAXIMUM ATTAINABLE SCORE = 8
SIMILARITY ATTRIBUTE = 6.15/8

FIGURE 7 Device selection example.

```
NAND-GATE: number of BITs
           number of devices in a package
           number of inputs
           output type
           ...

ADDER:     number of BITs
           number of devices in a package
           adder type (binary, BCD. . .)
           full or half
           enable type
           output type
           extra function
           ...

ALU:       number of BITs
           number of functions
           list of functions (complement, add multiply...)
           enable type
           output type
           ...

PROCESSOR: number of BITs
           number of instructions
           list of instructions
           number of arythmetic operations
           list of the arythmetic operations
           type of architecture
           number of address fields
           number of addressing types
           list of addressing types
           number of registers
           ...

           ...
```

FIGURE 8 Sample attribute sets of selected device classes.

edge encompasses the implicit and explicit restrictions based upon objects, operations, and relationships along with general and specific heuristics and inference procedures. For example, there are restrictions on the availability of a device in certain speed categories. Capturing these knowledge semantics is an important aspect of KB systems.

This approach to KB management provides much greater flexibility than the approaches which place the knowledge in a separate application program, or in a separate KB. The hyper-semantic data model provides for large data bases, a unified representation of knowledge and data, and mechanisms which allow abstract data typing.

Figure 8 shows the set of attributes for selected device classes. Each entity class has a restricted attribute set associated with it. This set is large enough that devices can be unique. Within the set of attributes for each class, each attribute has a restricted domain which is the set of values the attribute can contain. For example the off-the-shelf AND-gates will have only 2, 3, or 4-inputs. Typing of the attributes is enforced (i.e. attributes can take only values which are in the domain of the attribute). For each class of entities, each attribute is associated with a set of heuristics; from this set a designated one will be used at the evaluation of the similarity of the attribute.

Figure 9 presents an abbreviated knowledge/data schema for the KB. This figure shows the entity class of combinational circuits. The subclass ALU is an instance of the class combinational circuits. The specific ALU is an instance of the subclass ALU. The

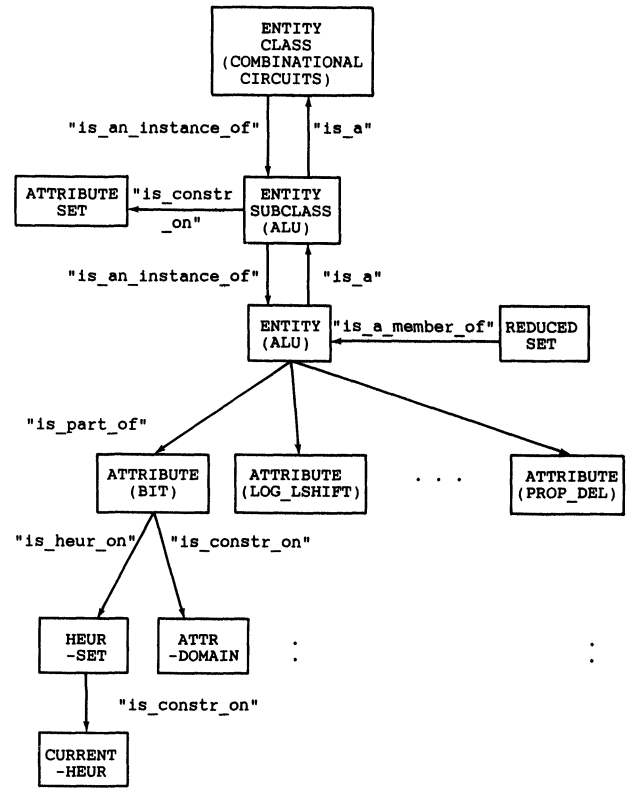


FIGURE 9 Abbreviated knowledge/data schema.

specific ALU may also be a member of the set of holograms which are selected for secondary attribute score processing. The specific ALU is described by an aggregate of attributes. These attributes belong to the attribute set valid for the subclass ALU. The schema also shows the set of heuristics associated with an attribute, from which the currently selected heuristic is used.

The KB will continuously change as new designs are incorporated, and new hardware functions become available. The attribute domains will expand, and so will the heuristics that evaluate them. Even when the attributes and heuristics change, the device selection process will continue the same way as is described in the section (*Similarity Assessment*).

The Actual Device Language

ADPs describe the transformation between the inputs and outputs of an HD. ADPs are expressed with Boolean equations, if-then-else statements and procedures. The ADPs also contain the signal identifiers of all ports to facilitate name unification.

ADL is not executed, however descriptions of HDs should be semantically correct and complete. It is important to contain all operations explicitly to allow for future expansion of the KB.

Figure 10 shows the ADPs for some combinational circuits: a buffer, a 5-input NOR-gate, and a 1-BIT full adder. Figure 11 presents ADPs for some sequential circuits: a bus buffer register, a 2-to-1 multiplexer with storage, and a random access memory (RAM). In Figure 11 stored values in latches or registers have a special notation. In the ADP of Figure 11(a) the previously stored value is denoted as being in the variable STORE. In Figure 11(b) a notation is added to express that STORE now has more than one location.

```
BEGIN
INPUT --> OUTPUT
END
```

(a) buffer

```
BEGIN
NOT(A1+A2+A3+A4+A5) --> B1
END
```

(b) 5-input NOR-gate

```
BEGIN
NOT Cin*NOT A*B + NOT Cin*A*NOT B +
        + Cin*A*B + Cin*NOT A*NOT B --> S

A*B + Cin*B + Cin*A                 --> Cout
END
```

(c) 1-BIT full adder

FIGURE 10 Sample ADPs for combinational circuits.

```
BEGIN
IF CLR THEN                                          L --> Q
      ELSE IF      (CLK AND (G1+G2))     THEN       ST --> Q
           ELSE IF (CLK AND NOT(G1+G2)) THEN D --> ST --> Q
           ELSE                                      ST --> Q
           END IF
END IF
IF (NOT OUTCONTROL1 AND NOT OUTCONTROL2) THEN Q --> OUTPUT
                                         ELSE Z --> OUTPUT
END IF
END
```

(a) bus buffer register

```
BEGIN
IF NOT SELECT THEN IF READ       THEN STORE(ADDR) --> DATA
                   ELSE IF WRITE THEN DATA --> STORE(ADDR)
                   END IF
              ELSE                              Z   --> DATA
END IF
END
```

(b) RAM

```
BEGIN
IF      (WSEL AND CLK)     THEN INPUT1 --> STORE --> OUTPUT
ELSE IF (NOT WSEL AND CLK)THEN INPUT2 --> STORE --> OUTPUT
ELSE                                      STORE --> OUTPUT
END IF
END
```

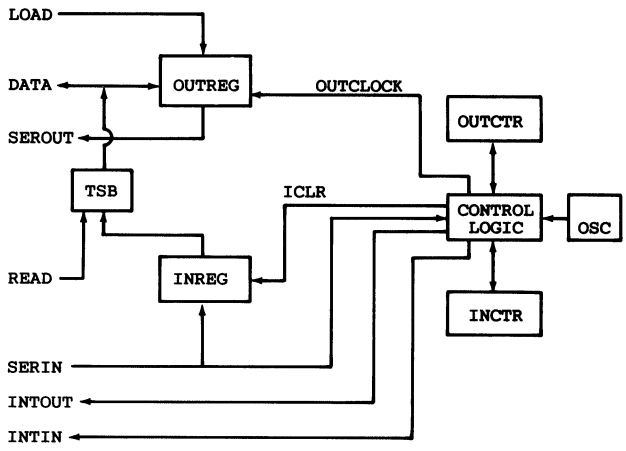
(c) 2-to-1 multiplexer with storage

FIGURE 11 ADPs for a bus buffer register, a RAM, and a 2-to-1 multiplexer with storage.

Figure 11(c) is an example of an HD having a clock input. This component has a clock input which is negative edge triggered; this fact is not contained in the ADP. However, the attribute NEGATIVE-EDGE-TRIGGERED will be included in the list of attributes for the module. In this figure, INPUT1, INPUT2, and OUTPUT imply 4-BIT buses, this information can be found in the type information for these signals, which is in the input, output, and local signal lists of the module.

When the hologram for Figure 11(c) is selected, the signal names in the ADP are unified with the signal names in the module. The module signal names take the place of WSEL, CLK, INPUT1, INPUT2, and OUTPUT signal names.

Procedures are used to describe more complex HDs: For example, expressing the operations of a Universal Asynchronous Receiver Transmitter (UART) in ADL is more involved. Figure 12(a) shows the internal block diagram of a UART. The functioning of this UART is partially given. DATA



(a) Internal block diagram.

```
BEGIN
    .
    IF LOAD THEN (DATA --> OUTREG  AND  INTOUT --> H)
    END IF
    L --> SEROUT
    FOR I = 7 TO 0 DO
        OUTREG(I) --> SEROUT
    H --> SEROUT
    .
    .
END
```

(b) Abbreviated ADP

FIGURE 12 UART internal block diagram and ADP.

input values are loaded into the output register OUTREG at the rising edge of the control input LOAD. After that, the contents of the OUTREG are transmitted to the SEROUT serially, starting from the highest BIT. The duration of the serial BIT time is controlled by the internal clock. When the output shifting process is complete, the interrupt output INTOUT is set low. Figure 12(b) shows a partial actual device primitive for the UART.

The actual device primitives combine to generate the function description of an intermediate module, as it is shown in the function description of Figure 4.

As a summary, ADPs have all the signal names to facilitate name unification and ADL is not executed. The function description of a module, together with attributes, local rules, local heuristics, and data regarding physical characteristics gives complete information for KB hologram selection and use.

VHDL TO IEI TRANSLATION

The details of the VHDL to the hologram representation translation, without attributes and function descriptions, can be found in [11]. Here it will be shown that the attributes and the function description can be also generated from the VHDL representation. Since an analyzer (compiler) is not available to generate the AST, the attributes and the function description will be generated from the VHDL representation itself.

As described in the section (OVERVIEW OF THE EXPERT SYSTEM), the hologram is a network of modules and signals. The IEI representation contains the root module and signals of the target hologram. This root module is named the 'specification module', since it contains all specification for the target hologram. The element assignment, netlist, and local slots of the specification module are empty, since until matching with KB holograms it is not known how a module is going to be decomposed. The signals contain type information, which will be needed in the HD recognition and hardware design processes.

The Function Description Generation

Figure 13 shows the function description of the specification module development process. The Function Representation Structure Generator (FRSG) trans-

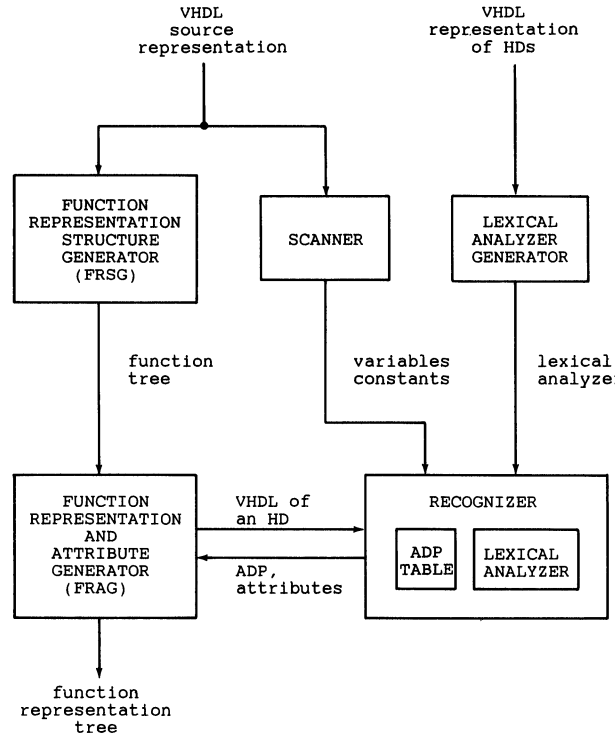


FIGURE 13 The function description of the specification module generation process.

forms the VHDL representation to the function tree. In VHDL, hardware systems are described in terms of components and interconnections between them. Hence the hardware design has a tree structure in which the components at the leaf nodes are behaviorally defined. The function tree corresponds to the tree structure of the VHDL representation.

This function tree is given to the Function Representation and Attribute Generator (FRAG) which transforms it to the function representation tree. The FRAG traverses the function tree and calls the recognizer for each leaf node. The recognizer, given the VHDL representation of an HD, recognizes the HD and its attributes, assembles the ADP and attributes, and presents them to the FRAG. The FRAG replaces the VHDL representation of the HD with the ADP and attributes. When all leaf nodes are translated, the function representation tree is complete. A pointer is placed in the function description slot of the specification module, which points to the tree just generated.

The scanner builds a table containing variables and their types, and constants with their values, and gives this table as an input to the recognizer. The scanner assembles this table from the declarations of the VHDL representation.

The set of HDs which have to be recognized will change frequently, each time a new design is incorporated in the KB. A lexical analyzer generator is included in the function description generation process to facilitate the incorporation of new designs. The lexical analyzer generator, given the VHDL representation of HDs (regular expressions), will generate a lexical analyzer which in turn will recognize HDs. This lexical analyzer is a major module in the recognizer. Examples of lexical analyzer generators are described in [17, 18].

```
entity MULTIPLEXER is
   generic (SERIES:STYP;FANIN,FANOUT:REAL);
   port    (WS,CLK  : in BIT;
            I1,I2   : in WORD;
            Q       : out WORD);
end  MULTIPLEXER;


architecture BEHAVIOR of MULTIPLEXER is
   process(CLK)
      begin
         if CLK = '0' and NOT CLK'STABLE then
            if WS = '1' then
               Q <= I1
            else
               Q <= I2
            end if
         end if
      end process
end BEHAVIOR;
```

FIGURE 14 VHDL representation for a 2-to-1 multiplexer with storage.

The Recognizer

As the most unique part of the AST to IEI translation the recognizer is described in greater detail. As was described earlier the recognizer translates the VHDL representation of an HD into the ADP representation and attribute set. The recognizer first recognizes the HD; second, it accesses the HD's ADP and the given attributes; third, it generates the actual signal names for the ADP from the VHDL representation; fourth, it generates the derived attributes; fifth, it returns the ADP and the attributes as the output.

The HD Recognition

Figures 14 and 15 show VHDL representations of HDs, a 2-to-1 multiplexer with storage, and a D-latch respectively. In Figure 14, when the signal CLK (clock) makes a positive transition, depending on the state of the signal WS (word select), signals I1 or I2 are latched into the multiplexer's storage. The content of the multiplexer's storage is presented on the output signal Q. In Figure 15, when signal EN (enable) is high signal D (data) is latched into storage, and when the signal EN is low, the content of the latch does not change. The content of the latch is presented on the output as signal Q.

The VHDL representations of HDs are regular expressions, since a Deterministic Finite Automaton (DFA) can be built which accepts them [19]. Figure 16 shows the DFA which accepts the VHDL representation of the 2-to-1 multiplexer. This automaton, upon receiving an input symbol advances to a next state. The input symbol is a token of VHDL. Identifiers are replaced with the appropriate words such as signal, variable, or constant to make the recognition general. This automaton is a Moore machine, because the final state indicates a particular HD recognized. Here HD-5 was recognized. Figure

```
entity D-LATCH is
   generic (SERIES:STYP;FANIN,FANOUT:REAL);
   port    (D,EN   : in BIT;
            Q,Q̄    : out BIT);
end  D-LATCH;


architecture BEHAVIOR of D-LATCH is
   process(EN)
      begin
         if EN = '1' then
            Q <= D;
            Q̄ <= D̄
         end if
      end process
end  BEHAVIOR;
```

FIGURE 15 VHDL representation for a D-latch.

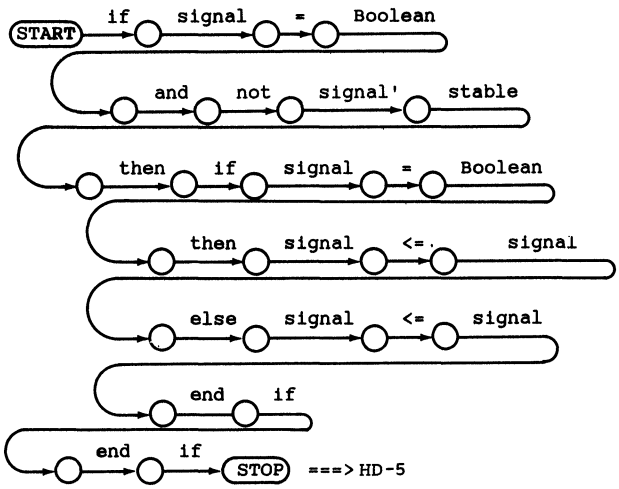


FIGURE 16 The DFA which accepts the VHDL representation of the 2-to-1 multiplexer.

17 illustrates the DFA which accepts the VHDL representation of HD-3, the D-latch.

The ADP and Attributes Generation

Once an HD is recognized, it is given an HD-number. The HD-number serves as an index into the ADP Table. The ADP table contains the following information for each HD-number:

a. The general ADP: in which the word "signal" stands for actual signal names. Figures 16 and 17 illustrate general ADPs.

b. A pointer to a procedure which translates the general ADP to a particular ADP. (That is each "signal" is replaced with an actual signal name.)

c. Given attributes: These are attributes which are not derived from the VHDL representation. For example, the attribute which places the HD in an HD class.

d. A pointer to a procedure which derives the rest of the attributes.

The above are different for each HD.

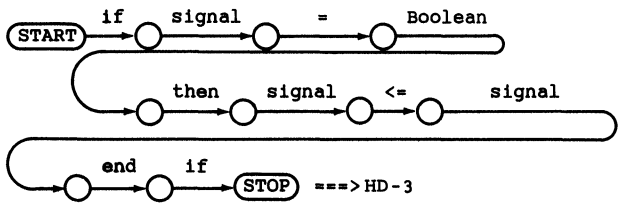


FIGURE 17 The DFA which accepts the VHDL representation of the D-latch.

The program which translates the ADP from its general to its particular form relies on positional placement of signal names in the VHDL representation. The order of the signal names in the VHDL representation is known along with the information as to which actual signal name replaces which "signal" in the ADP.

The attributes are either given in the ADP table or derived from the VHDL representation. As examples, let us follow the generation of the attributes for the 2-to-1 multiplexer and the D-latch.

Figure 18 presents the attribute set for the 2-to-1 multiplexer and the D-latch. In the ADP table, some of the attributes of these HDs are given. For example, for the multiplexer, the attributes 2-TO-1-MULTIPLEXER, and CLK are given; and for the D-latch the attribute, D-LATCH is given.

The program, which derives attributes from the VHDL representation, examines each segment in the VHDL representation. For the 2-to-1 multiplexer (Figure 14), the attribute NEGATIVE-EDGE is derived from the fact that in the VHDL representation, the segment "CLK = 'O' and NOT CLK'STABLE" means the clock just made a negative transition. Therefore, the multiplexer is negative edge triggered. So if the 'O' can be found in this phrase in the VHDL representation, the attribute NEGATIVE-EDGE is added to the attribute set. The 16-BIT attribute is derived from the fact that the type of signals I1, I2, and Q are 16-BIT BIT-vectors. The attribute TOTEM-POLE is also derived, since no resolution function is implied, and hence the output should be the standard totem-pole output. Other attributes such as SERIES, FANIN, and FANOUT are derived from the generic list of VHDL component instantiation. Default values are used if they are not specified. In the generic list LS, FANIN-1, FANOUT-10 were specified.

The attribute POSITIVE-ENABLE for the D-latch (Figure 15) is derived from the VHDL representation, since the enable EN should be '1' when the device is enabled, which means an active high enable. The attributes 16-BIT, TOTEM-POLE, LS,

```
key attribute: 2-TO-1-MULTIPLEXER
primary attributes: CLK, NEGATIVE-EDGE, 16-BIT
secondary attributes: TOTEM-POLE, LS, FANIN-1, FANOUT-10
```

a. Attribute set for a 2-to-1 multiplexer.

```
key attribute: D-LATCH
primary attributes: POSITIVE-ENABLE, 16-BIT
secondary attributes: TOTEM-POLE, LS, FANIN-1, FANOUT-5
```

b. Attribute set for a D-latch.

FIGURE 18 Attribute set for a 2-to-1 multiplexer and a D-latch.

FANIN-1, and FANOUT-5 are derived the same way as for the multiplexer.

Each VHDL representation is examined for additional knowledge; and if the additional knowledge is present, then the ADP is to be modified accordingly. For example, if one of the inputs is a constant (not a signal) to facilitate the design of this hardware, the ADP for this situation is modified. In the ADP the input signal name is replaced with the word "constant" plus the identifier of the constant. Other examples are knowledge of connecting signals to perform a function such as left shift, right shift, and dealing with partial buses.

IMPLEMENTATION AND PERFORMANCE

The VHDL Version 7.0 [3] was used and its translation to the hologram representation without attributes and function description is described in [11]. A subset of VHDL necessary for the design of the example problems was implemented. The details of the rest of the implementation of the hardware synthesizer can be found in [12]. The implementation of the hardware synthesizer was not exhaustive, but merely a prototype to identify issues which come up during hardware synthesis, and which are applications of the newly developed approach. In the VHDL to hologram translation presently the designer is helped by identifying the part of the VHDL representation that failed. As a later extension the designer could be provided with an explanation of why HDs and library modules were considered and rejected. In the initial hardware synthesizer 12 HDs were recognized, 26 KB holograms were implemented, and three test cases were designed.

A 16-BIT adder, a 16-BIT 6 operation ALU, and the bus structure of A Simple Computer (ASC) [20] were designed, modified, and evaluated. The design for the ALU contains 5 modules and 4 ICs. The design for the ALU contains 37 modules and 16 ICs. The design for the 3-bus structure of ASC contains 170 modules, and 70 ICs. Modifications of the designs demonstrated that existing designs from an earlier design cycle can be modified easily.

The designs produced by the hardware synthesizer are no different from the designs which would have been developed by a human designer. The reason for that is: first, some of the knowledge (the operations needed) was given in the VHDL representation; second, the knowledge necessary to design the test cases efficiently was available at the translation to the IEI process and in the inference engine.

CONCLUSIONS

The hardware synthesizer accomplishes several tasks: (a) recognizes HDs, (b) selects holograms from the KB, and (c) connects the components to generate a digital circuit.

HD recognition becomes particularly important when the KB contains a large number of similar components, since selecting and connecting HDs, when done manually, is a very time consuming and error prone process.

The limitation of the initial hardware synthesizer is that the designer's help is needed in the VHDL representation to hardware circuit translation. First, the designer's help is needed in the VHDL to IEI translation process when VHDL representations can not be translated to existing hardware devices. Second, when the design does not meet requirements, the VHDL representation must be modified by the designer by adding additional constraints. Therefore, it is not completely automatic.

When extensions fulfilling the above functions are incorporated into the hardware synthesizer, the design of a digital circuit will proceed automatically although the design may take several iterations.

Acknowledgment

The authors acknowledge the contributions of Ronald Green and Peter Klon to the initial phases of this research.

References

[1] S.G. Shiva, "Automatic Hardware Synthesis," *Proceedings of the IEEE,* Vol. 71, No. 1, January 1983, pp. 76–78.

[2] C.R. Green, Development of an Expert Hardware Synthesis System, Doctoral Dissertation, University of Alabama in Huntsville, AL. December 1985.

[3] VHDL Language Reference Manual, Version 7.0, IR-MD-045, Intermetrics, Bethesda, MD. January 1985.

[4] VHDL Tutorial for IEEE Standard 1076 VHDL, CAD Language Systems, Inc., Rockville, MD., May 1987.

[5] S.G. Shiva, and P.F. Klon, "The VHSIC Hardware Description Language", *VLSI Design,* June 1985, pp. 86–106.

[6] L.F. Saunders, "The IBM VHDL Design System", *Proceedings of the 24th ACM/IEEE Design Automation Conference,* 1987, pp. 484–490.

[7] P. Marwedel, "A New Synthesis Algorithm for the MIMOLA Software System", *Proceedings of the 23rd ACM/IEEE Design Automation Conference,* 1986, pp. 271–277.

[8] J. Bhasker, "Process-Graph Analyzer: A Front End Tool for VHDL Behavioural Synthesis", *Software-Practice and Experience,* May 1988, pp. 469–483.

[9] K.R. Muralidhar, and H.N. Mahabala, "Synthesis of a Control Unit from Instruction Set Specification in VHDL Environment," *VLSI Design '91 Digest of Papers 4th CSI/IEEE International Symposium on VLSI Design,* January 1991, pp. 200–205.

[10] C.R. Green, and S.G. Shiva, "PECOS—An Expert Hardware Synthesis System", *Proceedings of the Fifth International Workshop on Expert Systems and their Applications,* Avignon, France, May 1985, pp. 1295–1320.

[11] P.F. Klon, On Interfacing HDL to Knowledge Bases, Doctoral Dissertation, University of Alabama in Huntsville, AL. May 1986.

[12] J.U. Jones, Representation and Matching of Knowledge for the Design of Digital Systems, Doctoral Dissertation, University of Alabama in Huntsville, AL. March 1989.

[13] VHDL Support Environment Systems Specification, IR-MD-024-1, Intermetrics, Bethesda, MD. July 1984.

[14] P.H. Winston, Artificial Intelligence, Reading: Addison-Wesley, 1984.

[15] F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, Building Expert Systems, Reading: Addison-Wesley, 1983.

[16] A.M. Dewey, and S.W. Director, Principles of VLSI System Planning: A Framework for Conceptual Design, Boston/Dordrecht/London: Kluwer Academic Publishers, 1990.

[17] W.L. Johnson, J.H. Porter, S.I. Ackley, and D.T. Ross, "Automatic Generation of Efficient Lexical Analyzers Using Finite State Techniques," *Communication of the ACM,* December 1968, pp. 805–813.

[18] M.E. Lesk, "LEX—a Lexical Analyzer Generator," CSTR 39, Murray Hill: Bell Laboratories, 1975.

[19] J.E. Hopcroft, and J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Reading: Addison-Wesley, 1979.

[20] S.G. Shiva, Computer Design and Architecture, Boston: Little, Brown and Company, 1985.

Biographies

SAJJAN G. SHIVA is a Professor of Computer Science at the University of Alabama in Huntsville. He has conducted research in the areas of Hardware Description Languages and Automatic Synthesis under contracts from NASA, U.S. Army Research Office and Strategic Defense Command. He has authored two books: Computer Design and Architecture and Introduction to Logic Design, published by Harper Collins. His other areas of interest are in computer modeling and performance evaluation, distributed processing and software engineering in which he is a consultant to industry and government.

JUDIT UDVARHELYI JONES received MS in electrical engineering from the Technical University of Budapest, Hungary, and MS and PhD in computer science from the University of Alabama in Huntsville. She has 10 years of experience in electronic circuit design. She currently works for the Space Transportation Division of Rockwell International in Huntsville researching integrated health management systems. Her current interests include expert systems, languages, computer architecture.