DEVE: An Expert System for Hardware Design Verification

A. Mete Kabakçıoğlu1, Olga L. Mazuera2

1Department of Electrical and Computer Engineering, University of Miami, P.O. Box 248294, Coral Gables, FL 33124
2Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431-0991

Abstract: This paper describes an expert system approach to digital hardware design verification. Artificial Intelligence based approaches typically use general purpose theorem-proving to show that the design meets the formal specification. In contrast, the expert system DEVE presented in this paper interprets the specification to invoke proper domain specific verification methods in a knowledge-based environment.

1 Introduction
An important phase in any design process is verification, the task of showing that the design meets its specifications. A Prollog-based expert system, DEVE, has been developed for hardware design verification [6,7]. A formal hardware description language for the implementation and for the specification is the input to the verification system. The verification is achieved by interpreting the specification to invoke proper domain specific methods on the implementation model and by reasoning from the first principles. This expert system approach to hardware verification integrates formal and domain specific methods in a knowledge-based environment.

Artificial Intelligence based approaches to hardware verification commonly use logic as the formal framework for verification [2]. A formalized logic is used for hardware specification and representation. Then a theorem-prover, which mechanizes the given logic, can be used to prove formally that the design meets its specification [2,4,5]. However, general purpose theorem-provers are often inefficient and need human intervention. DEVE approach is based on the widely supported point of view that the use of suitable strategies for problem solving in the specific domain results in more efficient systems. Therefore, DEVE attempts to provide a verification tool in a knowledge-based framework by guiding the theorem-proving component provided by the Prollog interpreter with domain specific knowledge and methods. These include methods to derive behavior from structure such as conventional and symbolic simulation, minimization of Boolean expressions, binary decision diagrams (BDD) [1], backward (effect to cause) reasoning, knowledge about structural properties such as feedback, connectivity, behavior and structure of common hardware components such as gates, registers, ALUs and operator definitions which can be used in specifications. An important advantage of a knowledge-based organization is that new knowledge can be added incrementally.

2 DEVE
DEVE has two major components: The Domain Knowledge and the Verification Engine. The problem representation, the information provided by the user, consists of two elements: hardware specifications and queries expressed in a language named IRL, and the Design Structure of the hardware design. The task of DEVE is to prove that the given hardware meets its specifications. In order to express net relationships evolving over time a more powerful language than first-order logic is necessary. For that purpose the logic-based Invoke Reasoning Language (IRL) is developed for DEVE. IRL is interpreted by the Verification Engine which invokes proper reasoning mechanisms, hence its name.

The Design Structure of the hardware to be verified by DEVE is described as a set of interconnected components using Prolog facts. This representation is called extensional form [3]. This structural information, in conjunction with information in the Component Library about the behavior of primitive components, is used to derive the behavior of the described design. The Component Library, part of DEVE's Domain Knowledge, stores information about behavior of typical components such as elementary gates, flip-flops, multiplexers, decoders registers and ALUs. The components stored in this library constitute the set of primitives which can be used in the structural description. DEVE provides mechanisms to manage this library. The behavior of a new component can be defined in Prolog using definitional form [3]. This form of representation of hardware supports simulation. A macro expansion capability is also available to allow new components to be defined hierarchically in terms of available components. The behavior can be defined at different levels during the design process. For example a controller can be defined without going to the logic level of detail by describing its state diagram.

In IRL, specification language we make assertions about the nets of the design under consideration. A complete specification of the design is not necessary, i.e. partial specifications are possible. Different specified causal relationships between nets may involve functional relationships such as arithmetic/logic operations, causal
relationships with time dependency such as register transfers/operations on registers. The variables which represent the value of the nets in IRL are called netvariables. Each netvariable is associated with a netname, which is explicitly stated in IRL statements. Universal and existential quantification of netvariables is supported. The basic form of IRL statements is: IF conditions are true, THEN actions should occur. Conditions and actions are defined using the reasoning relations available in DEVE's Domain Knowledge. These reasoning relations support typical register transfer level operators which include arithmetic (add, increment, decrement, multiply), logic operations and data transfers.

An IRL IF-THEN statement used to specify a sequential design includes explicitly two arguments: current and next state names of the transition. The actions in the body of the then predicate of an IRL statement with state and next state names s1 and s2 respectively, are expected to occur only when the machine is in s1 and the conditions which cause a transition to s2 occur. The descriptions can be state driven or event driven. In state driven descriptions initially known values of memory elements such as registers is given. Next a series of IF-THEN statements show how the design evolves form the given state, i.e. expressing control flow and the corresponding data activity without need for explicit state encoding. Multiphase clock cycles are allowed and the elapsed time between s1 and s2 can be one cycle or several cycles. Event driven descriptions give all the conditions which cause the action to be verified.

IRL is a language embedded in Prolog. Therefore in the process of interpreting IRL, statements about sequential designs in the deductive domain of logic programming DEVE supports two levels of reasoning. In state driven specifications the first level of reasoning has the task of following the evolution of all execution paths of the IRL statements using the conditions in depth first fashion and at each state choose the proper verification strategy (such as simulation) on the implementation model. In event driven specifications the event constraints described by the conditions are used to choose the proper verification strategy. The second level involves the use of the results of the reasoning on the implementation model to prove corresponding actions. Typically there is a need to derive partial or complete behavior using the knowledge about the structure of the implementation model for verification. The derived behavior expresses different relationships between relevant nets of the design. Conventional logic simulation or symbolic simulation is used for that purpose. Symbolic simulation allows the verification for all possible cases, however the expressions can become very large to handle. BDDs (Binary Decision Diagrams) have been recently utilized to overcome this difficulty [1]. BDD's prove to be an efficient representation method for Boolean expressions and are employed by DEVE. Another method employed involves backward reasoning from cause to effect (i.e. from output to corresponding set of inputs). In addition, a novel method we developed for temporal reasoning on hardware based on time stamped symbolic simulation is available in DEVE [6,7].

An important aspect of an Expert System involves the explanation facilities. DEVE provides facilities for tracing the reasoning process at different levels of granularity. DEVE provides an additional tracing facility, which records as Prolog facts the input-output net values of design components evaluated during simulation or backward reasoning in response to an IF-THEN statement. The user can interactively query about input-output values of components to trace the causes that lead to a response on the implementation model.

Examples verified by DEVE include partial specifications of 16 & 32 bit microcode microprocessor design, and several combinational (arithmetic circuits) and sequential designs. We have also verified parameterized designs such as an N-bit adder, N-bit comparator and temporal specifications such as always, eventually about sequential designs [6,7].

It took two man weeks of work to model the implementation and specifications of a microprogrammed processor in DEVE. This is quite short in comparison of 10-12 months typically necessary for theorem-prover based verification [4,5]. After the ALU is proved to be correct, verification of instruction cycle is accomplished fully automatically on a Sparstation in the order of seconds whereas theorem-provers typically take time in the order of hours to days and need human intervention [2,4,5].

References