therefore perhaps limited by) the particular logic of the given proof environment. The research presented in this document makes use of a semantics that has been developed in an accepted semantic formalism without regard as to how it is to be mechanised. The generality of the target proof environment then allows one to seamlessly embed the semantics to reason not only about it but VHDL programs as well.

The most substantial piece of work in this field is the result of research at the Aerospace Corporation. The semantics of the simulation kernel have been implemented in the State Delta Verification System. SDVS is a framework for describing the execution of individual program steps based on the way that they affect the overall state in which the program executes [1]. The subset implemented is also large [17]. The idea is much the same as in the Siemens project (i.e. develop the meaning of a particular program based on a specification of the simulation model and then reason about it). Again, the contrast here is that the work is being carried out without recourse to an external oracle. The problem is not therefore in trusting both the oracle and the interface between two systems, but in gaining confidence in the formal specification of the VHDL simulation kernel.

Other similar research is just beginning at Computational Logic, Inc. [39]. The idea is much the same as in the Aerospace project. The system will be implemented in the Boyer-Moore logic [11] as realised in the NQTHM theorem-prover [12]. The Boyer-Moore system differs from both SDVS and the proof development environment that will be used later in that it is a much more automatic system, requiring less user intervention.

2.2.3 Other Approaches

Research into verification environments for VHDL is not limited to those methodologies given in Sections 2.2.1 and 2.2.2. Other projects make use of either custom-designed or existing formal languages for their implementations. One is underway at Royal Holloway and Bedford New College where a language called FUNNEL is being developed [34]. The language is intended as a descriptive medium for many different hardware description languages, and therefore provide a framework for reasoning about designs irrespective of the language in which they were specified. Another approach is being taken at IRISA [7] using the language SIGNAL to describe the simulation kernel of VHDL and to rationalise its timing model.

here. The work in Cincinnati has lead to the development of valuation functions for initialisation and state transformation during a simulation. There have, however, been no proofs of properties of the semantics to emerge from the effort as of yet.

2.2.2 Operational Work

Research closer in flavour to that presented here is underway at several sites. These projects focus on an operational formalisation of what is specified in the Language Reference Manual. Even within this unanimity of spirit, the approaches differ in practice. One type of formalisation is based on extra-logical manipulation of the semantics which passes the results to a proof assistant for additional manipulation. The other does all the work in whatever logical system has been chosen.

Use of an Oracle

Research at Siemens [35] is characteristic of the first approach. The simulation model of VHDL is written in the functional language ML. The results of working with a program inside this specification of the simulator are then passed to the LAMBDA system for reasoning. LAMBDA is a proof assistant from Abstract Hardware, Ltd. that is based on the same logic as the system used here. The difference is that it has been specially tailored for reasoning about hardware. In essence, the ML program is being used as an oracle providing a statement of behaviour for the theorem-prover to use in subsequent proofs. The goal of the Siemens project has been to provide an industrial-strength tool, and the subset of the language is therefore very large. The approach chosen, while not strictly formal in that the semantics of the language is specified by an interpreter written in a functional language rather than securely inside the logic of LAMBDA, has merits in that it provides an efficient environment for working with VHDL programs. The limitation is that one must be convinced of the correctness of the external oracle before trusting any results that arise from it and the subsequent proof process.

Secure Formalisations

The other approach, namely that of modelling the semantics of VHDL inside the mechanisation of the particular logical system being used, is much closer in flavour to the work that follows in later chapters. The important difference between it and the material summarised here is that the formalisations are expressed in (and are

particular program. These assertions could then be embedded in the text of the program for use during simulation in much the same way as VAL specifications. The notion was that once the behaviour of the canonical version of a device had been agreed upon, a statement describing its behaviour could be extracted automatically and embedded in any other program purporting to be the same device. Any exception raised during simulation would therefore indicate some divergence from the accepted norm which needed to be investigated.

Both these projects were based on the respective researcher's interpretation of the semantics of VHDL. No formal statement about the semantics of the language was available at the time beyond that given in the Language Reference Manual. Furthermore, they relied on traditional simulation practice. While this was not a bad thing in that it brought a degree of rigour to the examination of the simulation model of VHDL, these research efforts did not result in any definitive mathematical statement about the semantics of the language.

2.2 Semantics-Based Verification

More recent research has been directed at the verification, rather than validation, of VHDL programs. To accomplish their goals, all the projects involved in this area have necessarily had to build up a formal notion of the semantics of VHDL. The research efforts currently under way vary not only in their approaches to the particular problem of semantics, but also in their implementation of it. Furthermore, they are characteristic of emerging international interest in the field. One camp bases their approach on primarily denotational methods. The other is grounded in a more dynamic intuition.

2.2.1 Denotational Work

Early work in discerning a semantics for signal attributes emerged from IMAG in Grenoble [33]. The emphasis was to statically understand the simulation behaviour associated with the attributes of individual signals. The work eventually led to a prototype system for reasoning about a subset of VHDL similar to that chosen here. A drawback was that because of its insistence on a unit-time model, 0-delay signal assignments could pose problems in the framework.

Promising early work on a full denotational description of the VHDL simulation kernel is underway at the University of Cincinnati [16]. The research is again based upon a small subset of VHDL that is not dissimilar to that presented

CHAPTER 2

RELATED WORK

Efforts at applying formal methods to VHDL programming fall into two categories. The first deals strictly with increased support for simulation-time validation of programs. The second pertains to the search for and use of a tractable semantics for a reasonable subset of the language. The rest of this chapter presents a broad overview of these approaches.

2.1 Simulation-Time Validation

The late 1980's was a period of great activity in research surrounding the development of simulation-based validation tools. This particular type of system was meant to provide the designer with additional assurance that a given program was behaving in the intended way through clever use of the simulator itself. Because it followed shortly after the introduction of VHDL as a standard in the United States, the research into these methods was carried out there.

One of these investigations led to the development of the VHDL Annotation Language (VAL) at Stanford [4,5]. VAL allowed the user to decorate a program with specifications about its operation. These annotations appeared as comments in the VHDL program text, and were expanded by a pre-processor before the program was compiled and simulated. When the program was executed, the annotations triggered warning messages if the design differed in its execution from the implicit specification that they provided. This particular methodology was not only a way of getting more useful information out of a simulation, but also made it easy for clear specifications of the behaviour of various parts of a design to be embedded in program texts.

Another early project, conducted by the author, attempted to automatically generate VHDL assertions characterising the behaviour of programs [36,37]. The philosophy was much the same as in the VAL effort. The difference was that it was based on automatically ascertaining the characteristic statement of behaviour for a